Indoor Path Planning Using Magnetic Positioning

Daniel Herrero

School of Electrical Engineering

Espoo 20.06.2016

Thesis supervisor:

Prof. Simo Särkkä

Thesis advisor:

M.Sc. (Tech.) Ville Tolvanen

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
FINAL PROJECT

Author: Daniel Herrero

Title: Indoor Path Planning Using Magnetic Positioning

Date: 20.06.2016 Language: English Number of pages: 12+68

Department of Electrical Engineering and Automation

Professorship: Sensor informatics and medical technology

Supervisor: Prof. Simo Särkkä

Advisor: M.Sc. (Tech.) Ville Tolvanen

Precise positioning in indoor environments faces different challenges than outdoor ones. The attenuation of satellite signal indoors has triggered the development of a wide range of indoor positioning systems. There are many different approaches to successfully locate users or objects in indoor environments like hospitals, universities or shopping centers. Most of them uses radio frequency or ultrasonic systems. In this thesis, the indoor user's location has been determined based on the unique magnetic field created inside buildings. Using this magnetic positioning, there is no extra hardware implementation required, reducing the development and maintenance budget. Given this position, a full indoor guidance system has been created and implemented in a mall, computing the route from user's position to the desired goal with a path planning algorithm based on Voronoi nodes. This system has been implemented on an Android application where the user can choose any indoor goal, by a list of preset possible goals, or any outdoor place. When the current position and goal are known, the path planning algorithm computes the best route and it is displayed on the smartphone. This work has been joined in a bigger project where indoor and outdoor guidance were combined, so the final application is able to guide the user both indoors and outdoors. The indoor-outdoor transition detection has been made by another partner and has also been integrated into the final app. In conclusion, the aim of this thesis has been to prove magnetic positioning as a valid tracking system in a fully indoor and outdoor guidance system developed for Android.

Keywords: Indoor positioning, guidance, A*, Voronoi, path planning, magnetic field, GPS, Android

Preface

First of all, I would like to thank Professor Simo Särkkä for his guidance and help during the whole project. His assistance has been remarkably helpful with the scope and goals of this thesis, as well as the content and style of the final document. Furthermore, he has shown his interest and time during these months of work.

I would also want to thank Ville Tolvanen because he has been actively involved in this project, helping with every issue related with IndoorAtlas software. His support and advices regarding the mapping part have been really useful.

I am also grateful to IndoorAtlas company who let me use their system and example codes to develop my own code. This project would not have been possible without the provided SDK and the example Android app.

My partner of the project, Juanjo, has also helped and encouraged me during the development of the thesis, so I am also thankful to him. Moreover, we worked together to build the final application and test it for debugging. I do not want to forget my family and friends who have supported and advised me as much as they could. The final result is also thanks to them.

Finally, I am really thankful to Aalto University for letting me do my Master Thesis with their facilities and resources, specially Niina Huovinen from Student Services Office, who has explained me all the procedures required to complete a Master Thesis in Aalto University.

Otaniemi, 20.06.2016

Daniel Herrero

Contents

List of Figures

Abbreviations

AGPS	Assisted Global Positioning System
AOA	Angle of Arrival
AP	Access Point
BFS	Breadth-First Search
BS	Base Station
CDT	Constrained Delaunay Triangulation
DFS	Depth-First Search
DGPS	Differential Global Positioning System
FIFO	First Input First Output
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IPS	Indoor Positioning System
MS	Mobile Station
RSSI	Received Signal Strength Indicator
SDK	Software Development Kit
WLAN	Wireless Local Area Network

1 Introduction

Modern buildings are becoming bigger and more complex and many times people get lost inside. Public buildings like universities, hospitals or train stations are examples of places where people do not know how to go to a certain place. In this thesis, an **indoor guidance** system has been developed and tested in a small shopping mall.

The problem of navigation can be summarized into three simple questions: *"Where am I?"*, *"Where do I want to go?"* and *"How should I get there?"*. The first question is called localization, given a known or unknown environment how I can work to obtain my position, based on what I see or what I have seen before. For the second question you need to specify a goal, and then, the third question solves the problem of path finding, which basically has to provide a path that results in achieving the desired goal (Leonard & Durrant-Whyte, 1991).

The problem of position determination has been always considered. Nowadays, outdoor location is mainly done by GPS (Global Positioning System), which performs really well in open areas (Grewal, Weill, & Andrews, 2007). However, when the user is inside a building the GPS signal is attenuated with the structure materials and location can not be obtained by this technology.

Unfortunately, localization inside buildings remains not robust for precise positioning. Typical indoor environments contain multiple walls and obstacles made of different materials, so a unique indoor positioning system is not suitable for all cases. It has led to the development of many different approaches that try to locate and track objects within buildings and closed areas, such as radio frequency or ultrasonic systems. They try to locate a traveler or an object by measuring distances, angles or time of flight of a specific signal to some fixed receivers (Koyuncu & Yang, 2010).

A distinct approach has been taken in this thesis. The location has been obtained based on **magnetic field**. The Earth's magnetic field interacts with the metallic structure of the building, creating a unique field that can be used as a map for indoor positioning. The main advantage of this system is that no hardware components are required to be installed, so there is no cost of implementation or maintenance of the system. The sensors needed for measuring magnetic field are available in most of modern smartphones. This technology has been developed by the company IndoorAtlas (Haverinen, 2015) (Haverinen & Kemppainen, 2009). In order to be able to accurately position the user inside a building with this technology, the magnetic field in the area must have been previously mapped. Once it has been properly mapped, user's position is provided with an accuracy of 1-2 meters.

Regarding the second question of *"Where do I want to go?"*, the possible goals inside a mall may be the shops, restaurants, services or any other facility of it. Once the list of possible goals have been done, each one has to be located as a particular point of the map.

Now that the position and the goal are known, the third question shows up. Path planning needs to provide a path to go from your position to the desired place. There are many algorithms, but not all of them are suitable for this project. The provided path is going to be displayed to the user and she is supposed to follow it.

There are some algorithms that just provide a feasible path, which does not have to be the shortest or the fastest one (Bohlin & Kavraki, 2000). However, if the user wants to go to a restaurant, for example, she would like to follow the shortest path. Moreover, the provided path has to be natural, avoiding going too close to walls or obstacles.

Concerning all these constrains, different algorithms have been implemented and compared to obtain the best one for our purpose. Finally, the algorithm was based on Voronoi nodes to maximize the distance from the path to the obstacles (the provided path in a corridor will be in the middle of it, instead of close to one side) (Kavraki, Švestka, Latombe, & Overmars, 1996). A graph was created with these Voronoi nodes, and the path must follow these nodes. In order to find the set of nodes that configure the path, the A* algorithm was implemented due to its good performance and its really low execution time (Yao, Lin, Xie, Wang, & Hung, 2010).

All this theory has been successfully implemented on an Android application, where the indoor guidance is made. Furthermore, the final application is also able to provide outdoor guidance and detects indoor-outdoor transition thanks to the work of a co-worker. The user can be guided both indoor and outdoor with the same application, commuting from one mode to the other automatically thanks to the indoor-outdoor transition detection.

2 Background

2.1 Path planning

2.1.1 Introduction

Planning is a term that means different things to different groups of people, but all of them need to have algorithms that convert high-level specifications of task from humans into low-level descriptions of how to move. The terms *motion planning* and *trajectory planning* are often used for these kinds of problems. A classical version of motion planning is sometimes referred to as the *Piano Mover's Problem*. Given the map of a house, the algorithm must determine how to move a piano from one room to another avoiding collisions. Robot motion planning usually ignores dynamics and differential constrains and focuses on the translations and rotations required to move the piano. Recent work, however, does consider other aspects such as uncertainties, modeling errors and optimality. Trajectory planning usually refers to the problem of taking the solution from a robot motion planning algorithm and determining how to move along the solution in a way that respects the mechanical limitations of the robot (LaValle, 2006).

Some questions that could come up to your mind would be, What is a plan? How is the plan represented? How is it computed? How good is a trajectory? Who is going to use that plan? This chapter provides general answers to these questions. Regarding user plan, it clearly depends on the application. In most applications, an algorithm would execute the plan; however, the user could also be a human. In that case the planning algorithm provides you the route to go from your position to a desired place.

Basic features of planning

There are several features and terms that are going to be used a lot in the following chapters, so it would be useful to define them clearly (LaValle, 2006).

- **State**. Planning problems involve a *state space* that contains all the possible situations that could arise. The *state* could represent the position of a robot, the location of the tiles in a puzzle or the velocity of a vehicle. Both discrete and continuous state spaces will be allowed. The state space is usually represented *implicitly* by a planning algorithm. In most of the applications the state space is too large to be explicitly represented. Nevertheless, the definition of the state spaces is an important component in the formulation of a planning problem and in the design of algorithms that solve it.

- **Time**. All planning problems involve a sequence of decisions that must be applied over time. Time might be explicitly modeled, as in a problem such as driving a car as quickly as possible. On the other hand, time may be implicit, by simply reflecting the fact that actions must follow in succession, such as in the case of solving a puzzle. Another example of implicit time would be a solution to the Piano Mover's Problem, where the actions can be converted into an animation over time. As in the case of state spaces, time can be either discrete or continuous.

- **Actions**. A plan generates actions for each state that could drive you to a different state. Considering the Rubik's cube as an example, given a state (a configuration of the tiles) you may execute an action, which could be rotating one of the sides, that will manipulate the original state and may end in a new one. For some problems, actions would be pretty easy to define, but other times they might not be trivial.

- **Initial and goal states**. Planning problems usually start with an initial state and some actions will be executed in order to achieve the goal state. The actions should be selected in a way to make this happen.

- **A criterion**. There are generally two different kinds of planning concerns based on the criterion:

    1. **Feasibility**: Find a plan that causes arrival at a goal state, regardless the efficiency.
    2. **Optimality**: Find a feasible plan that optimizes performances in some way. An example could be arriving to the goal as soon as possible, so time will be minimized, or finding the shortest path between the origin and the goal.

- **A plan**. In this context, a plan may simply specify a sequence of actions to be taken. If the problem is more complicated, the plan can also specify actions as a function of the state. In this case, regardless of the future states, the appropriate action is determined.

What is an algorithm?

There are different definitions of algorithm, but the mathematical definition is (Ng & Subrahmanian, 1992):

> A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

The idea is that the algorithm will provide you some rules that define a sequence of operations. For example, when children are taught how to divide, the rules and steps needed are a simple algorithm. Regarding to planning algorithms, they solve the problem of calculate the path between the origin and the goal given the criterion. These rules given by the algorithm must work in any case, in order to find the set of actions that leads you to the goal (Moschovakis, 2001). There are a lot of different algorithms. Some of them are really simple and cost efficient, and some other much more complex but give better results. The mosts important ones will be covered further on.

Problem formulation

Path planning models used throughout this thesis will be defined using state-space models. Most of them will be natural extensions of the model presented in this section. The basic idea is that each distinct situation for the world is called a *state*, denoted by x, and the set of all the possible states is called a *state space*, X. The state space should be large enough to have all the relevant information needed to solve the problem, but having irrelevant information can easily convert a simple task in a really difficult one (LaValle, 2006).

The current state may be changed through the applications of *actions* that are chosen by the planner. Each action, u, when it is applied from the current state, x, produces a new state, x', specified by a *state transition function*, f. The equation that represent this idea is:

$$x' = f(x, u). \tag{1}$$

The *action space* for each state is represented by $U(x)$, which represents all the possible actions that could be taken from x. The same action may be applied from different states, so it is convenient to define the set U of all the possible actions over all states:

$$U = \bigcup_{x \in X} U(x). \tag{2}$$

In addition, the *initial state* x_I and the *goal state* x_G or a *set of goal states* X_G must be defined. The task of a path planning algorithm is to find a finite sequence of actions that when applied, transforms the initial state x_I to some state in X_G.

The model could be summarized in the formulation 2.1.1:

Formulation 2.1.1

1. A nonempty *state space* X, with a finite number of states.
2. A finite *action space* $U(x)$ for each state $x \in X$.

3. A *state transition function* f that produces a state $f(x, u) \in X$ for every $x \in X$ and $u \in U(x)$. The next state will be provided by the *state transition equation* $x' = f(x, u)$.

4. An *initial state* $x_I \in X$.

5. A *goal set* $X_G \subset X$.

2.1.2 Basic algorithms

In this section the most basic planning algorithms will be covered.

Depth-first search (DFS)
Depth-first search, or DFS, is a way to traverse the map. This algorithm is basically applied to simple graphs, but it covers the main principles of graph theory. The idea of DFS is pretty simple: to go forward (in depth) while there is such possibility, otherwise to backtrack (Tarjan, 1972). For each node, it will look for all possible successors and check if any of them is the goal. Otherwise, it will search again for more nodes deeper in the tree and repeat again the same process.

A pseudocode for a simple DFS would be (Yan & Han, 2002):

DFS Pseudocode

```
// Simple DFS
void dfs (node position){
    mark this node as visited
    for (each successor adjacent to current node)
        if (successor is visited)
            skip it;
        if (next is the goal node)
            stop the search;
        else
            dfs(successor)
    }
}
```

This algorithm will always find a path from the origin to the goal, but it does not have to be the optimal one. This kind of a blind search, which does not take into account the location of the goal, is likely to take much more time than an intelligent search.

Dijkstra's algorithm
Dijkstra algorithm is an algorithm for finding the shortest paths between nodes in a graph. There are many variants; the original one (Dijkstra, 1959) just finds the

shortest paths between two nodes, but a more common variant fixes a single node as the source and finds the shortest paths from this node to all other nodes in the graph, providing a shortest-path tree. It can also be used for finding the shortest path between a single node and a single destination by stopping the program once it is determined.

As a result, this algorithm is widely used in network routing protocols, to find the shortest route between one city and all other cities (Misa & Frana, 2010).

Dijkstra algorithm can be resumed in the following steps:

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

2. Set the initial node as current and mark all the other nodes as unvisited.

3. For the current node, consider all unvisited neighbors and calculate their tentative distances. Compare the new distance with the assigned one and set the lowest value.

4. When all the neighbors have been considered, mark the current node as visited.

5. If the destination node has been marked visited (the path have been found when planning a route between two points) or if the smallest tentative distance among nodes in the unvisited set is infinity (there is no connection between the initial node and remaining unvisited nodes), then stop.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current node, and go back to step 3.

Here this algorithm will be explained with a basic example. Consider the graph given in Figure 1 where the starting node is 1 and the goal node 5. Taking the origin node as the current node, we compute the distance to all unvisited neighbors (Figure 1a) and mark *node 1* as visited. On the next step, we take the *node 2* and repeat the same process. As *node 3* has already a distance, we compare the two distances, through *node 2* or directly from *node 1* and keep the lowest one (Figure 1b and 1c). We repeat the same process now with the unvisited node with the lowest distance, *node 3*, and repeat the same process and update the assigned node distances (Figure 1d). In Figure 1e we reach the goal, and the rest of unvisited nodes, just *node 4* in our case, have a higher or the same distance than the one we got to arrive to the goal. In this scenario, the algorithm is finished. Otherwise, we would have to keep going with the unvisited nodes with lower distance than the goal, to explore if there is a shorter path than the one we found.

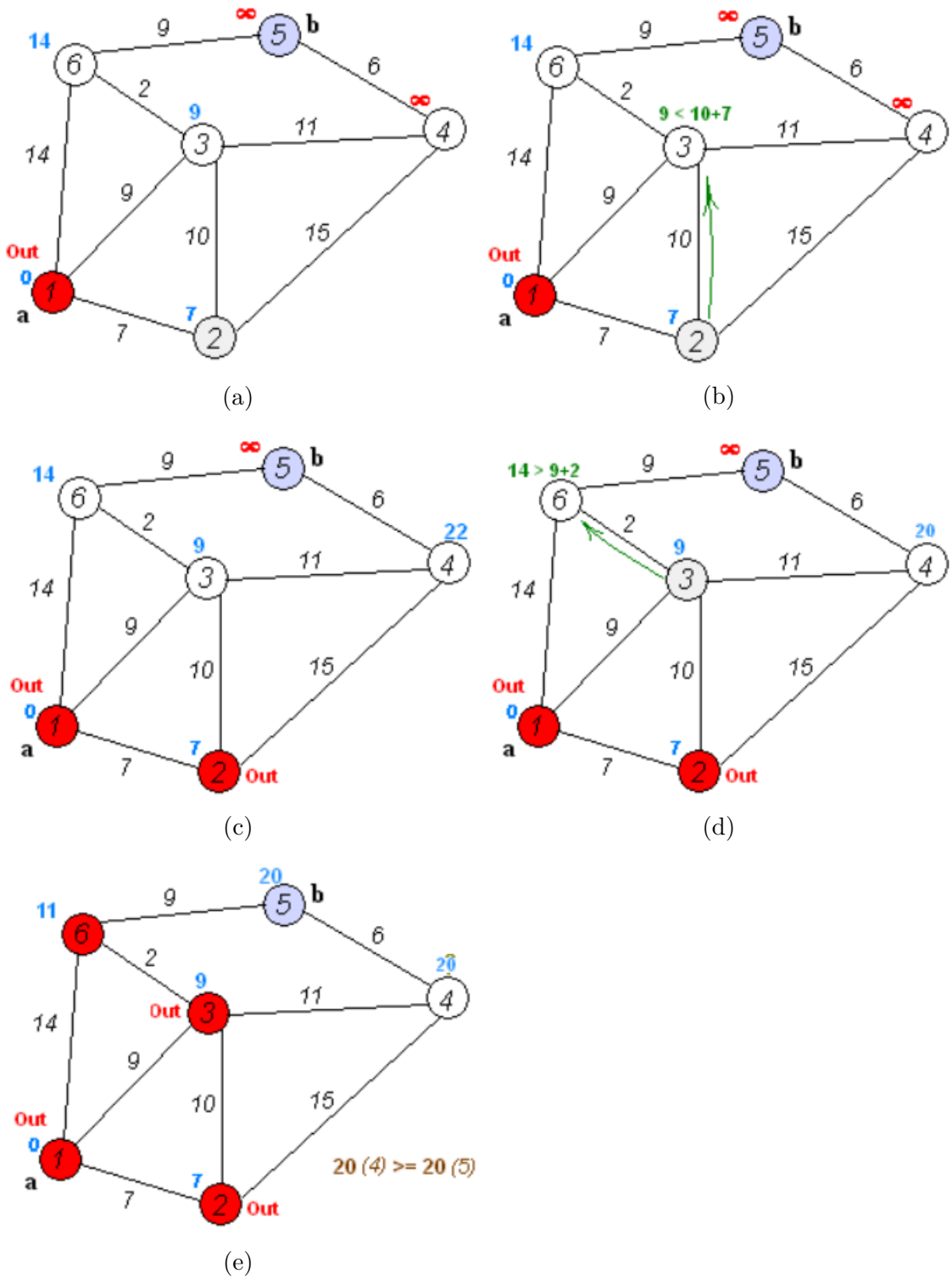


Figure 1: Dijkstra example (source: `https://en.wikipedia.org/wiki/Dijkstra's_algorithm`)

Breadth First Search (BFS)
The breadth first search is an algorithm that not only guarantees a path to the goal, but also it will provide the shortest one (Zhou & Hansen, 2006). In this method, we examine each of the nodes next to the node in question before we go deeper. This had the effect of forming concentric circles around the nodes we are searching. Since we are moving out from the center, we are going to reach the goal as quickly as possible. This method is marginally harder to implement, but it is still easy. BFS it is not recursive since we have to look at all the options before going deeper (Korf, 1985).

The pseudocode it is shown below:

BFS Pseudocode

```
// Simple bfs
struct node {
    position pos;
    node *parent;
}

void bfs (node start_position){
    add start_position to the queue
    while (the queue is not empty){
        item=pop a node off the queue;
        mark item as visited;
        generate all the successors to item;
        set the parent of each successor to item;
        for each successor
            if (the sucessor is the goal)
                end the search;
            else
                push it to the back of the queue;
    }
}

if (we have a goal node)
    find the path (node->parent->parent->...);
else
    there is no path;
```

This algorithm performs better than Dijkstra. We still can optimize the search in order to find a path more quickly. Breadth-first search can be viewed as a special-case of Dijkstra's algorithm on unweighted graphs, where the priority queue degenerates into a First Input First Output (FIFO) queue (Felner, 2011).

Potential field

The potential field algorithm can be explained as the behavior of a small ball rolling in a hill. The ball will roll down from the starting point towards the lowest point (Hwang & Ahuja, 1992). If we assume that the lowest point is our goal, we can create a potential field that models the forces the ball would suffer (Koren & Borenstein, 1991). Figure 2 illustrates this concept (Safadi, 2007).

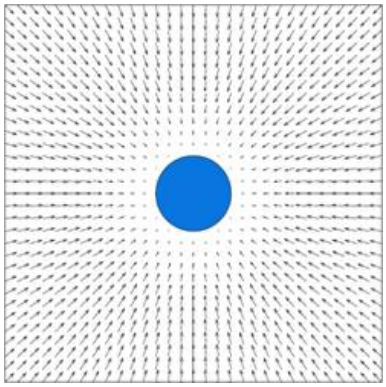


Figure 2: Potential field distribution of the goal (source: `http://www.cs.mcgill.ca/~hsafad/robotics/`)

We can also define the opposite behavior that allows the ball (or the robot) avoid obstacles. In this case, we simply make each obstacle generate a repulsive field around it. If the robot approaches the obstacle, a repulsive force will be applied, pushing the robot going away from it (Figure 3).

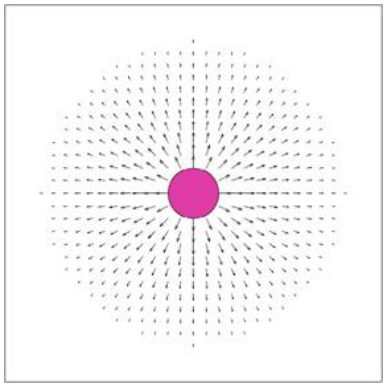


Figure 3: Potential field distribution of an obstacle (source: `http://www.cs.mcgill.ca/~hsafad/robotics/`)

If we combine this two components in the same field, we will obtain a field similar as the one represented in Figure 4 and Figure 5. The gradient points towards the direction of the maximum variation of the potential field. Taking again the example of the ball on the hill, the gradient will point the most leading slope from your

current position. The path will be obtained by applying the gradient to your current location in the field (Figure 6).

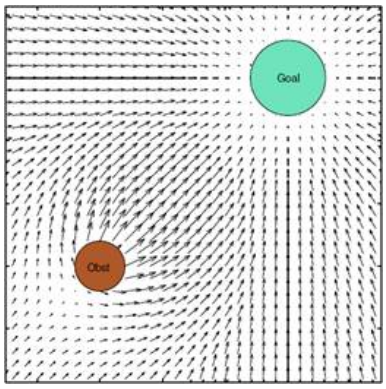


Figure 4: Complete potential field distribution (source: `http://www.cs.mcgill.ca/~hsafad/robotics/`)

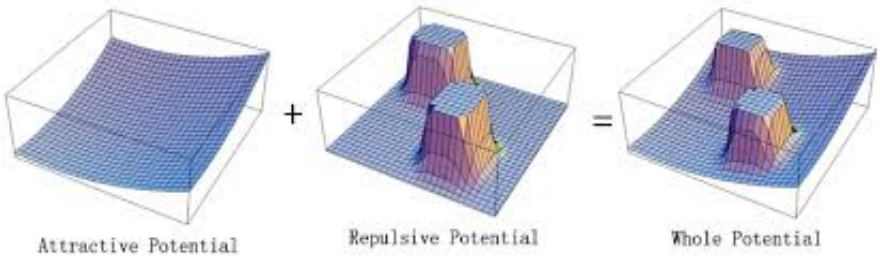


Figure 5: Combined view of the effects of obstacles (source: `https://taylorwang.wordpress.com`)

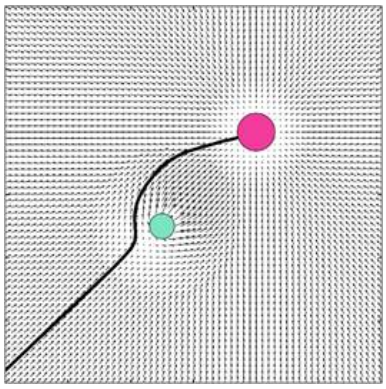


Figure 6: Path obtained with an potential field algorithm (source: `http://www.cs.mcgill.ca/~hsafad/robotics/`)

2.1.3 A*

The A* algorithm is also a generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored, if additional information is available that provides a lower bound on the distance to the target (Mehlhorn & Sanders, 2008). A* is an informed search algorithm, meaning that it solves by searching among all possible paths to the solution for the one that has the smallest cost.

This algorithm is noted by its good general performance, providing better results compared to other algorithms as is explained in Delling, Sanders, Schultes, & Wagner, 2009. However, it has been proved that there are others with better performance in travel-route systems (Zeng & Church, 2009). At each iteration of its main loop, A* determines which of its partial paths will be explored more. It is done based on an estimated *cost to go* to the goal node. Specifically, A* selects the path that minimizes

$$f(n) = g(n) + h(n) \tag{3}$$

where n is the current node, $g(n)$ is the cost it took to get to the node from the start, and $h(n)$ is a heuristic estimation of the cost of the cheapest path from n to the goal. A* will find the best path in a very short time provided your h is perfect. However, we cant determine h perfectly without doing other path finding so we just use an approximation. This estimation is often computed as the euclidean distance between these two points, which will not be the real distance if there are any obstacles, but it will work pretty well.

The pseudocode uses two list, **OPEN**, which contains all nodes that have already been visited but not expanded (successors have not been explored yet), and **CLOSED**, with the nodes that have been visited and expanded.

This can also be explained with the example given in Figure 7. The origin is represented by a green circle and the goal by a blue one. The nodes that have been visited are marked in orange. In the first step (Figure 7a), we have to choose between the node a and d. We compute and compare $f(a)$ and $f(d)$. As it has been explained before,$f(a) = g(a) + h(a)$, where $g(a)$ is the cost it took to arrive to the node a from the origin, in this case 1.5, and $h(a)$ is the cost to go, computed as the euclidean distance from a to the goal, 4. The resultant $f(a)$ and $f(d)$ are 5.5 and 6.5, respectively, so node a will be chosen, and we can move to the next iteration. Node a will be marked as visited and node b is added to the **OPEN** list. Repeating the process with the current nodes contained in the list (b and d) the graph is being explored, until we reach the goal.

A* Pseudocode

```
move node_start to the OPEN list
initialize the node as: f(node_start)=h(node_start)

while OPEN list is not empty {
    node_current= node from the OPEN list with the lowest f
    if node_current is node_goal
        we have found the solution
    generate node_successors from node_current
    for each node_successor{
        successor_current_cost= g(node_current) + distance
        between node_successor and node_current
        if node_successor is in the OPEN list{
            if g(node_successor)< successor_current_cost
                skip successor;
            }
            else{
                if node_successor is in the CLOSED list
                    if g(node_successor)< successor_current_cost
                        skip successor;
                    eliminate node_successor from the CLOSED list
                    add node_successor to the OPEN list
                else
                    add node_successor to the OPEN list
                    h(node_successor)= euclidean distance between
                    node_successor and node_goal
            }
        g(node_successor)=successor_current_cost;
        set the partent of node_successor to node_current
        f(node_successor)=h(node_successor)+g(node_successor)
    }
    Move node_current from the OPEN list to the CLOSED list
}
```

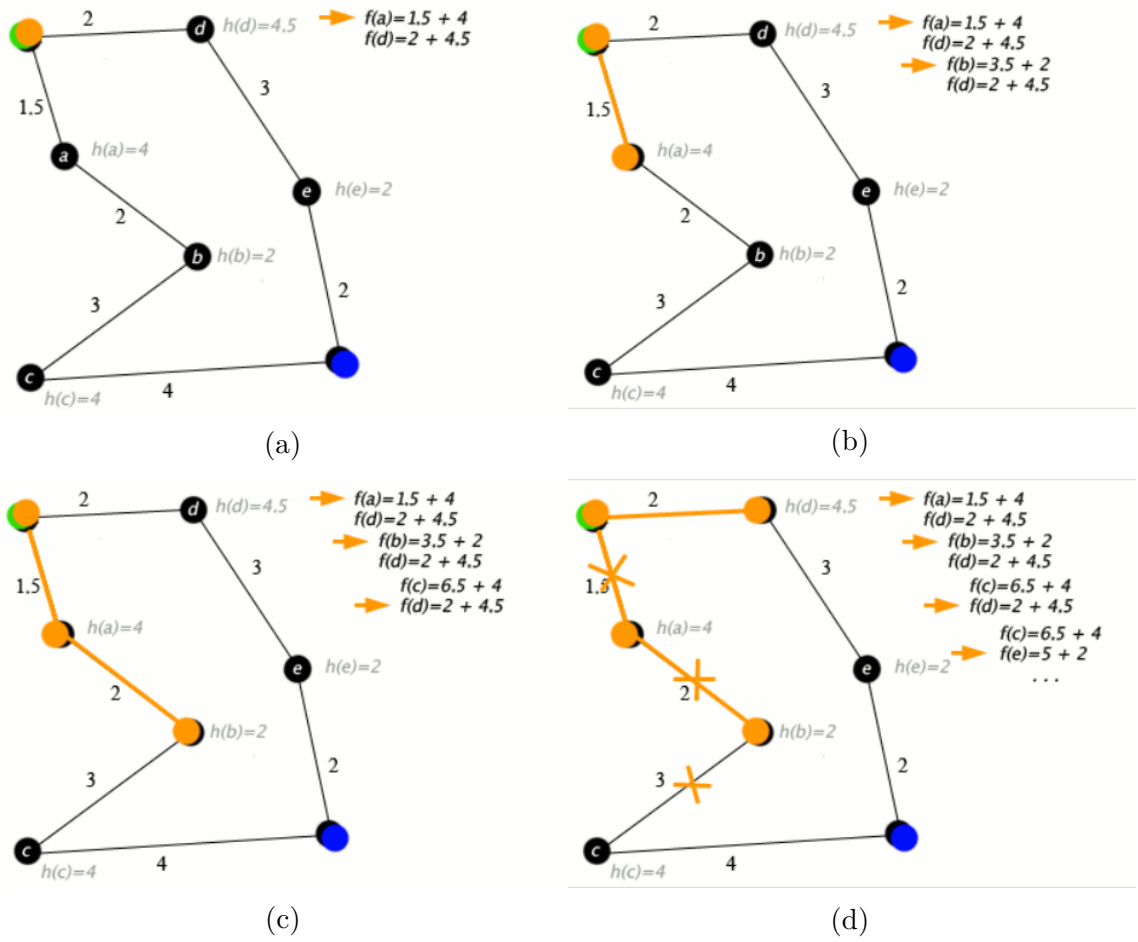


Figure 7: A* example (source: `https://en.wikipedia.org/wiki/A*_search_algorithm`)

2.1.4 Voronoi diagram and Delaunay triangulation

Voronoi diagram is a partitioning of a plane into areas based on the distance to certain points called Voronoi nodes. For each node there is a corresponding region consisting of all points closer to that node than to any other. These regions are also called Voronoi cells. An example of a Voronoi diagram is shown in Figure 8.

Voronoi diagram can be applied to many fields, like astrophysics, epidemiology, architecture or autonomous robot navigation (Yap, 1987). Related to the last one, Voronoi diagrams are used to find clear routes, trying to avoid collisions. If we define a node for each obstacle, Voronoi diagram will give us some cells. The edges of the Voronoi cells will provide us routes that will maximize the distance to the obstacles. Taking this into account, we can create some safe paths that will avoid, as much as possible, the obstacles.

The dual of the Voronoi diagram, the *Delaunay triangulation*, is the unique trian-

gulation so that the circumsphere of every triangle contains no other nodes in its interior (Fortune, 1992). These triangles are formed by three Voronoi nodes whose circumcircle is empty and a *Delaunay edge* connects two nodes that have an empty circumcircle (Figure 9).

There is a clear relation between the Voronoi diagram and the Delaunay triangulation. The Voronoi nodes correspond to the center of the circumcircles created by the Delaunay triangulation, as it is shown in Figure 10. If no four nodes are co-circular then the Delaunay triangulation is unique.

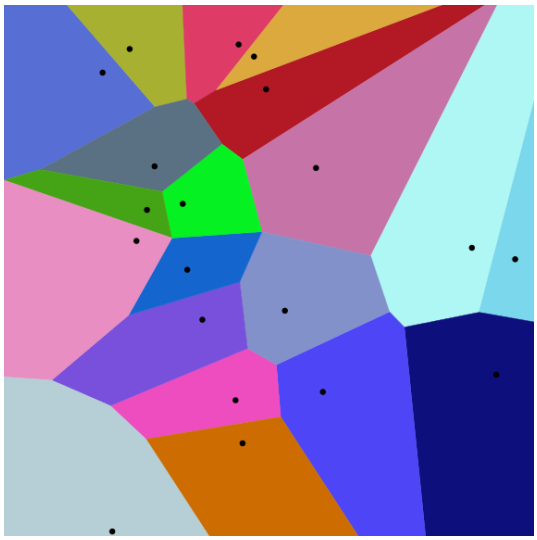


Figure 8: 20 points and their Voronoi cells (source: `https://en.wikipedia.org/wiki/Voronoi_diagram`)

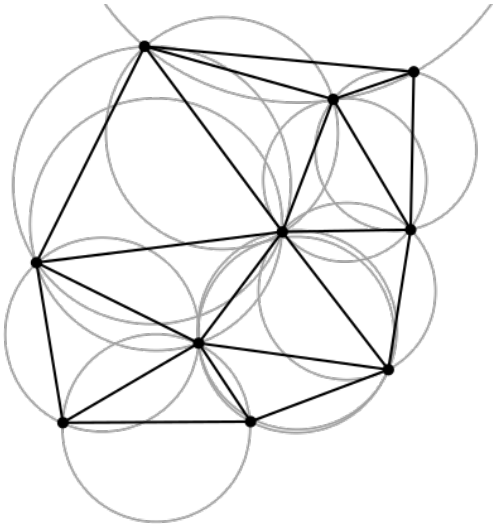
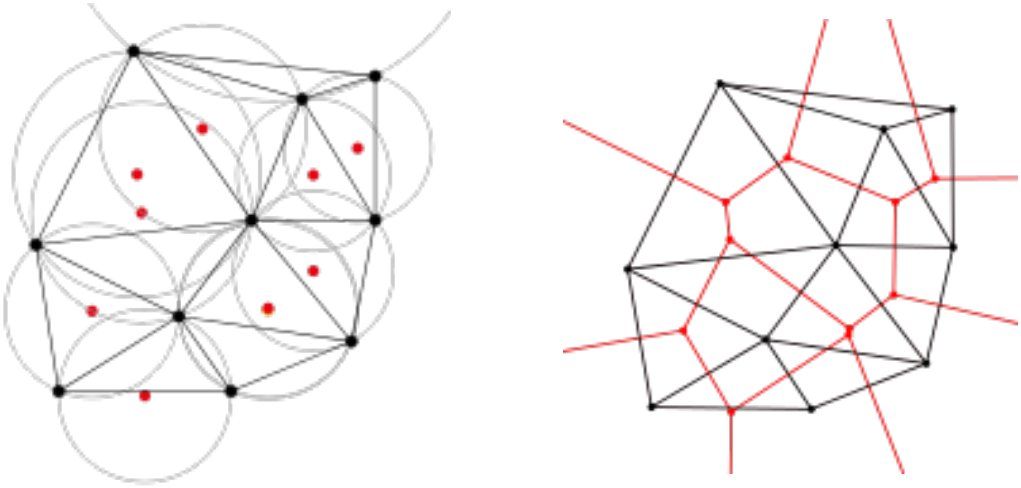


Figure 9: A Delaunay triangulation in the plane with circumcircles shown (source: `https://en.wikipedia.org/wiki/Delaunay_triangulation`)



(a) (b)

Figure 10: Relation between Delaunay and Voronoi. The Delaunay triangulation with all the circumcircles and their centers in red 10a. Connecting the centers of the circumcircles the Voronoi diagram is created 10b (source: `https://en.wikipedia.org/wiki/Delaunay_triangulation`)

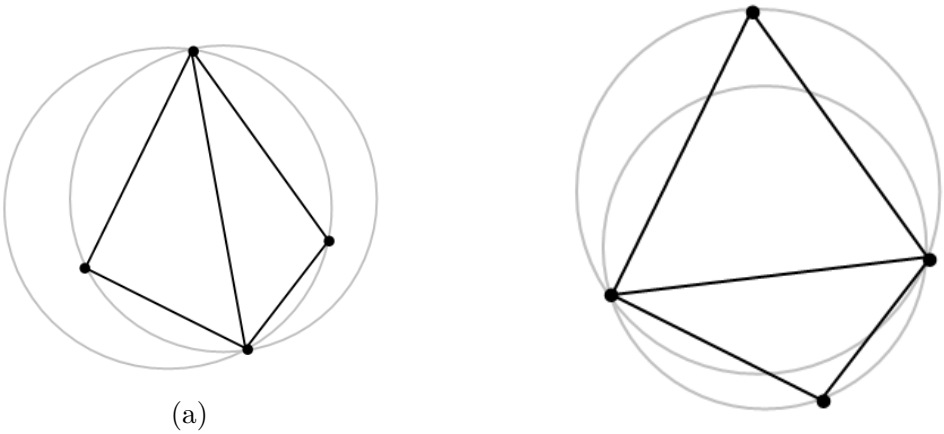
The constrained Delaunay triangulation (CDT) would be the same as the unconstrained one if there is no constrains. If we add some constrains, we force that line, or lines, to be part of the edges of the set of triangles generated. The CDT can be used to find a path between points, making sure not to cross some edges of the graph (Chew, 1989).

Some algorithms to compute the Delaunay triangulation are:

1. *Flip algorithms.* These kind of algorithms triangulate the points, and then they will check if that triangulation created meets the Delaunay conditions. Otherwise, the triangulation flips, as it can be seen in Figure 11 (de Berg, Cheong, van Kreveld, & Overmars, 2008).

2. *Incremental.* Repeatedly one vertex is added at a time, retriangulating the affected parts of the graph. When we add a new vertex, we split in three the triangle that contains this vertex, and then we apply the flip algorithm (Guibas, Knuth, & Sharir, 1992).

3. *Divide and conquer.* In this algorithm is also explained in de Berg et al., 2008. We draw a line to split the vertices into two sets. The Delaunay triangulation is computed for each set, and we merge the two sets along the splitting line. The merge operation can be done in time $O(n)$, so the total running time is $O(nlog(n))$



(a)

(b)

Figure 11: The circumcircles of 11a contains more than three points. Flipping the common edge produces a Delaunay triangulation of the four points (11b) (source: `https://en.wikipedia.org/wiki/Delaunay_triangulation`)

2.2 Indoor positioning systems

Indoor positioning systems locate and track objects within the buildings and closed areas. There are many different approaches like wireless concepts, optical tracking or ultrasonic techniques. Most of the existing solutions are based on trilateration methods using some references, like beacons, labels or tags (Koyuncu & Yang, 2010).

These systems are used to locate people or required objects in large buildings, for example, locating patients on a hospital or finding people in public areas as airports.

As mentioned in Section 1, localization inside buildings do no provide the same results as GPS does outdoors. There are many different indoor positioning systems that try to solve this problem, based on different technologies. Some of the current positioning systems are compared by their range and accuracy (Mautz, 2009).

2.2.1 Indoor vs Outdoor

Positioning is becoming increasingly important in today's society, and so many applications need to know your position to give you more detailed information. For instance, Google Now provides you interesting information to the user based on where you are, like near restaurants or the time needed to go back home.

Nowadays, GPS is the most popular positioning technology in the world and it is widely used for these purposes because of its accuracy and cost. However, GPS signal received may be extremely weak and not reliable in many everyday environments

such as indoors or urban areas. In these places, the building blocks the line-of-sight to GPS satellites (Barnes et al., 2002).

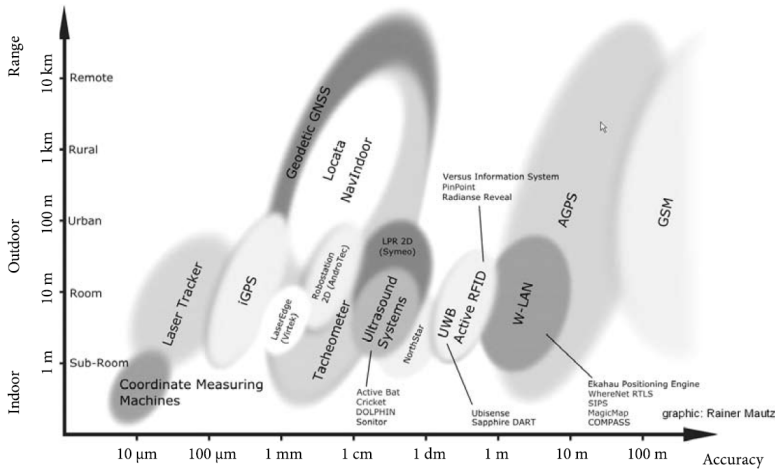


Figure 12: Positioning systems according to their accuracy and coverage area (Mautz, 2009)

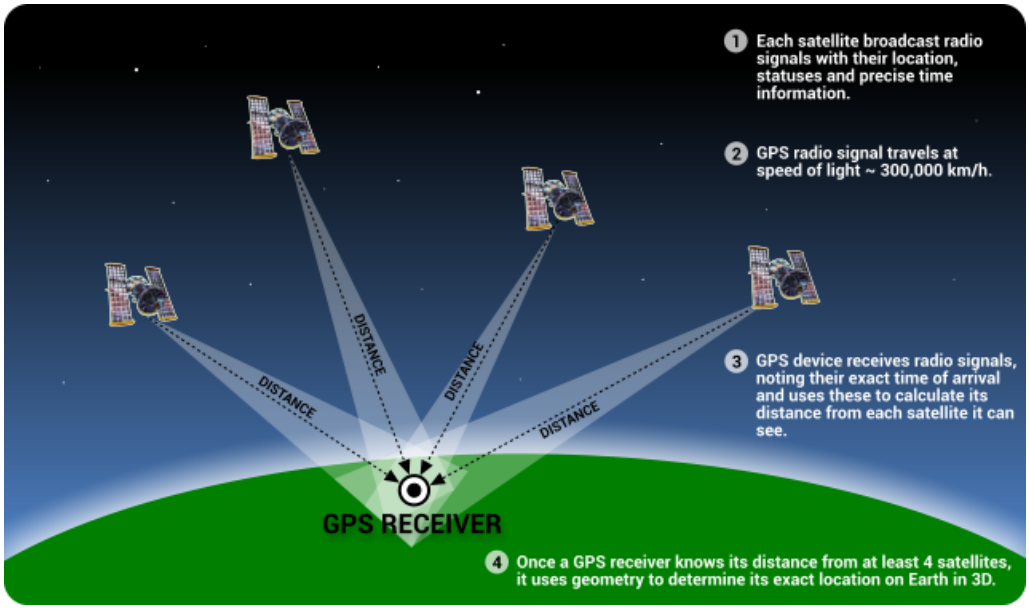


Figure 13: Working principle of Global Positioning System (source: `http://itsabouttimebook.com/how-gps-works/`)

Regarding indoor positioning, new emerging technologies have been developed for fixing that problem. Two totally different approaches have been taken. The first of them attempts to increase the sensitivity of the GPS receivers that try to detect very weak signals. On the other hand, some technologies use signals provided from other

systems not designed for positioning, like Wi-Fi, mobile phone networks or television (Zeimpekis, Giaglis, & Lekakos, 2002). Using these signals, we can compute the distance to some routers, for which we know the location, and estimate the position by trilateration.

These solutions are really attractive because they solve the main problem of GPS signal using existing infrastructure, but often their performance is not as good as the GPS with a clear view of the sky in terms of accuracy, reliability and simplicity. However, **High Sensitivity GPS** can provide positioning in some indoor locations. Even though the signal is attenuated with some materials, it could be used. Nonetheless, the accuracy reached is really low, so other approaches should be considered (Zandbergen & Barbeau, 2011).

Standard GNSS (Global Navigation Satellite System) receivers do not perform good enough in indoor environments because the weak signals from the satellites are barely undetectable. The signals are attenuated around 20-30 dB, it means that they are reduced 100-1000 times. **AGPS** (Assisted GPS) solve this problem using differential corrections and other information directly obtained from the the GNSS satellites (Djuknic & Richton, 2001). This weak signal can be used to obtain a improved position by integration over multiple intervals, so longer acquisition times are required. The AGPS accuracy indoors is enough to know in which room the user is. If higher accuracy levels are needed, it can be improved by DGPS methods (Differential Global Positioning Systems) (Kirkpatrick et al., 2002).

2.2.2 Radio frequency systems

Bluetooth Beacons

Localization using beacons may be achieved using some different techniques:

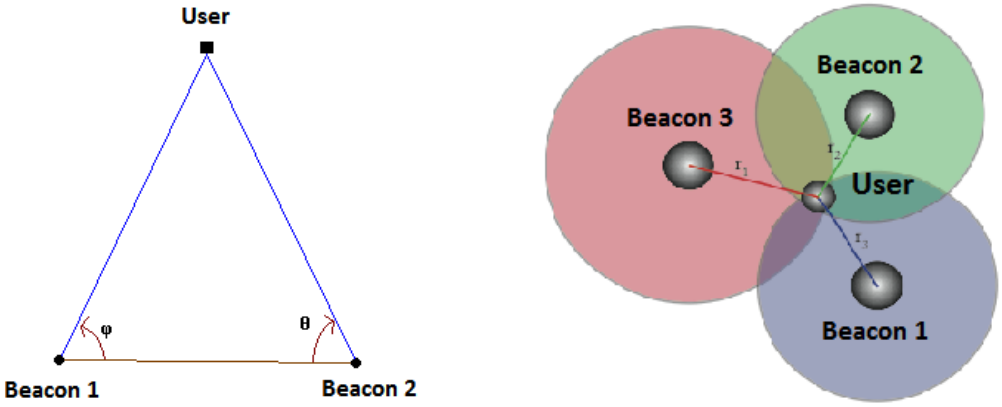
1. **Triangulation**. It only involves **angle** measurements between the line connecting two beacons and the line of sight from each beacon to the user. In Figure 14a the position of the user is computed by the angles ϕ and θ.

2. **Trilateration** requires the computation of the **distance** of the user from each of three beacon. The traveler's position is determined by the unique intersection point of three circles centered at each beacon, with a radius like the distance to the traveler 14b.

3. **Multilateration** systems determine the position of the user by measuring **time intervals** between the transmission of a pulse from the user and its reception at multiple receivers. This idea is applied in 2.2.3.

4. **Cell-based** methods obtain the user's location based on only the **visibility** of beacons, without using any angle or distance. Knowing the visibility range of the beacons, we can determine in which region the user is. The accuracy is

given by the number and position of the beacons, so both parameters will be key factors on the final performance 15.



(a) Triangulation

(b) Trilateration

Figure 14: Comparison between triangulation and trilateration for positioning (source: `http://www.wikiwand.com/fr/Triangulation` and `http://mattdonahoe.com/springs/`)

There are some approaches similar to WLAN that measures the received signal strength indicators (RSSIs) like a distance from the device. As signal is attenuated with distance, the higher intensity the closer you are to the device. However, the short range compared to WLAN has some disadvantages, like the number of devices that need to be deployed to provide an adequate coverage. Consequently, the cost of the whole system is high. Previous work on that field have proved that the results are not encouraging (Bielawa, 2005) (Hallberg, Nilsson, & Synnes, 2003).

Alternatively, another approach uses only visibility of Bluetooth beacons instead of RSSI. A simple boolean variable indicates if a certain beacon is in range or not. By using a large number of inexpensive beacons, the position of the user can be determined. Localization is done using a cell-based method that provides the region of intersection of visible beacon (Chawathe, 2009). In the example given in Figure 15, the traveler is in *line of sight* of B, C and D, but not in range of A and E. Consequently the only region where the user may be is in the colored one.

If we assume that we already know where the beacons are placed and the visibility of each of them, then it is not hard to know the region location of the user. However, where to put the beacons and how many of them install it is not obvious. Intuitively, the more beacon the system has, the better accuracy you will obtain. The number of beacons installed is usually fixed by the budget. Another key factor is the *interesting locations* of your floorplan, which may be, e.g. the most visited areas of a building. The position and range of each beacon will be determined in such a way that every

interesting location has a unique set of visible and non-visible beacons (Chawathe, 2008).

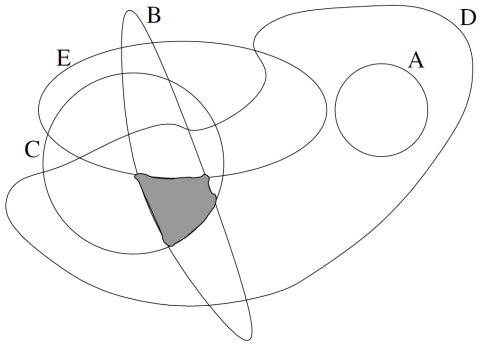


Figure 15: The principle of Active Bat location system (Chawathe, 2009)

WLAN
The deployment of WLAN is increasing considerably for delivering web services in limited areas like schools, universities, malls, museums and other public places. This development provides a lot of router points that can be useful for indoor positioning. There are some projects that try to use this infrastructure itself for positioning (Prasithsangaree, Krishnamurthy, & Chrysanthis, 2002).

The main **advantages** of WLAN-based localization are (Muthukrishnan, Koprinkov, Meratnia, & Lijding, 2006):

1. No hardware required: the wireless network is already deploy in many public areas like museums, universities or shopping malls. Therefore, location can completely be done by software methods.

2. Range: compared with other technologies like RFID or bluetooth, the range covered is bigger, reaching 50-100 m.

3. No line of sight required: WLAN systems are nor restricted to line of sight, meanwhile IR need it.

On the other hand, the main problem is to accurately measure the distance from an Access Point, due to complex signal propagation.

In order to achieve the position, the distance to these fixed points needs to be determined by either measuring the signal strength or the propagation delay (Bill, Cap, Kofahl, & Mundt, 2009).

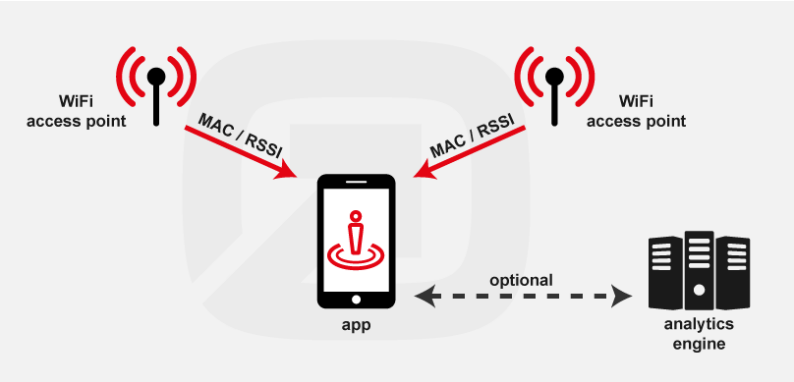


Figure 16: WLAN based positioning system (source: `http://www.infsoft.com/blog/2015/indoor-navigation-using-wifi-as-a-positioning-technology`)

There are some existing commercially available applications like EKAHAU [1] positioning engine. This system uses small, long-lasting, battery-powered Wi-Fi tags that are attached to the tracked assets. Existing Wi-Fi devices, like phones,tablets or laptops, can also be tracked. Once you access to the Wi-Fi Access Points (APs), the algorithm calculate the accurate position of these devices. The location accuracy can be highly enhanced by using Location Beacons. This system uses location "fingerprinting" to accurately predict and display in real time asset and staff location (Kaemarungsi & Krishnamurthy, 2004).

2.2.3 Ultrasonic positioning systems

Active bats With this system, the user needs to wear small badges that transmit an ultrasonic pulse. Measuring the time of flight from the badges to a grid of fixed sensors placed on the area where you want to be located. By a multilateration algorithm, the system is able to provide each tag's position with an accuracy of approximately 3 cm (Hazas & Hopper, 2006). However, Active Bat requires a large number of precisely positioned ultrasonic receivers. The principle of operation is shown in Figure 17.

[1] www.ekahau.com

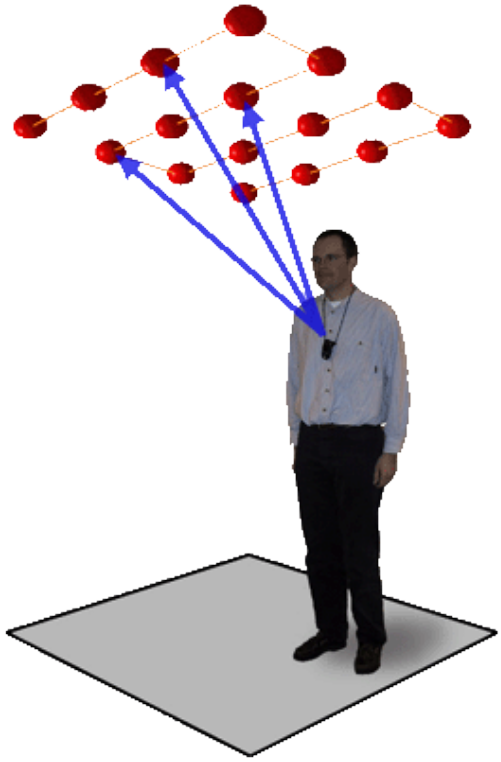


Figure 17: The principle of Active Bat location system (source: `http://www.cl.cam.ac.uk/research/dtg/attarchive/bat/`)

Crickets Cricket nodes are tiny ultrasonic devices than can work as transmitter and receiver. They have been developed by MIT and they reach an accuracy of 1-2 cm. However, the environment should be less than $10m^3$ (Priyantha, 2005). The appearance of these devices can be seen in Figure 18.

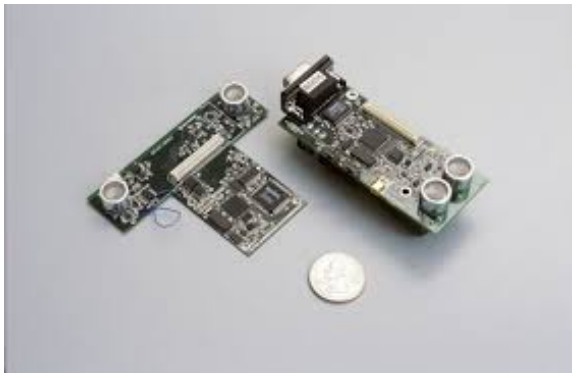


Figure 18: Cricket ultrasonic device (source: `http://cricket.csail.mit.edu/`)

Dolphin This ultrasonic system stands for Distributed Object Locating System for Physical space Inter networking and consists of distributed wireless sensor nodes (see Figure 19). They send and receive ultrasonic signals and, using a distributed positioning algorithm, is able to locate the objects with an accuracy of 2 cm. Again, the environment it works is quite small, only on a 3 by 3 meters square (Fukuju, Minami, Morikawa, & Aoyama, 2003) (Minami et al., 2004).

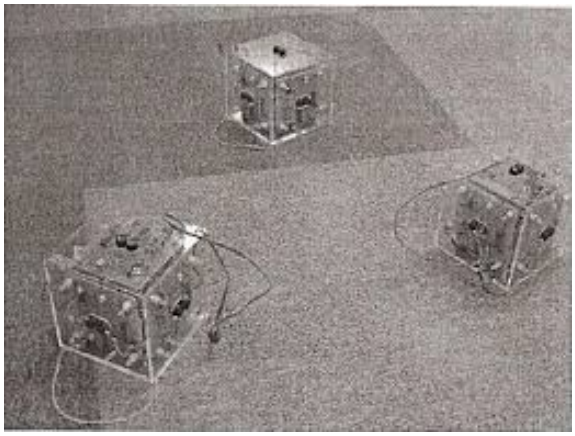


Figure 19: Dolphin ultrasonic system (Minami et al., 2004)

2.2.4 Magnetic field

Another different approach to indoor location is based on the magnetic field. This technique uses the metal structure of the building. The Earth's magnetic field will interfere with this metal components, so a unique magnetic field is created inside each building. The systems based on *magnetic field positioning* map the building by recording the magnetic field value in some determined points, and then the user will be located comparing these data with the actual one (Li, Gallagher, Dempster, & Rizos, 2012) (Storms, Shockley, & Raquet, 2010). This technology is really cost efficient because you do not need to install beacons as reference points, or buy expensive sensors to read magnetic field data. All new smart phones include a magnetic sensor, so a simple mobile phone can be used as a receptor.

There are some existing magnetic field based systems such as *IndoorAtlas*. They have developed a really easy system for mapping your own buildings and locating yourself. More information about this technique can be found in their web page `https://www.indooratlas.com` or in Section 3.2. In that Section, it will be explained how this system has been used for mapping and positioning in a building.

Comparing this technology with Wi-Fi and Beacons, the geomagnetic technology provides better accuracy and it does not require maintenance of large costly infrastructures (Haverinen & Kemppainen, 2009). This technology achieves 1-2 meter accuracy meanwhile Radio-based technologies, like Wi-Fi the accuracy level reached

has been 4 meters (Bahl & Padmanabhan, 2000). Furthermore, to have this accuracy consistently in Wi-Fi additional access points need to be installed, which can be technically challenging due to the rapidly changing radio environments.

The places where geomagnetic technology may be useful would be very different depending on the purpose. In some big public places such as train stations, this system can be used to guide people to reach their goal, for instance. This idea can also be export to museums, universities or hospitals. This system also provides real time tracking of a person, so it can be utilized as well for this purpose.

2.2.5 Cellular networks

Mobile localization has received considerable attention over the past few years. The basic function of a location system is to gather information to provide the position of a mobile station (MS) working in a specific geographical area. A popular approach, called *radiolocation*, measures parameters of radio signals that travel between a MS and a set of fixed stations, known as base stations (BS). Actually, the GPS location system is based on that principle (Liu, Darabi, Banerjee, & Liu, 2007).

Some other alternatives have been recently taken, such as cellular network. These approaches employ a cellular network as the transport mechanism for relaying the location estimate. Reading the signal measurements, like cell strength or length and/or direction of the signal, the MS position is computed with geometric relationships (Riter & McCoy, 1977).

Signal strength radiolocation is a well known location method that uses a mathematical model describing the path loss attenuation with distance (Figel, Shepherd, & Trammell, 1969). Measuring the signal strength, you can estimate the distance between the MS and the corresponding BS. By using multiple BSs, the global position can be easily determined by triangulation (Figure 20). For this approach, the main source of error is fading and shadowing, which can produce huge variations in the signal strength (Hellebrandt, Mathar, & Scheibenbogen, 1997).

Another different technique utilize the **angle of arrival (AOA)** as a measure to estimate the MS location. AOA is defined as the angle between the propagation direction of an incident wave and some reference direction, which is known as orientation (Singh, Shakya, & Singh, 2015). One common approach to obtain AOA measurements is to use an antenna array on each sensor node (Peng & Sichitiu, 2006).

The orientations of the unknowns may or may not be known at the time of deployment. Anyway, the localization under both scenarios can be solved by triangulation (Figure 21).

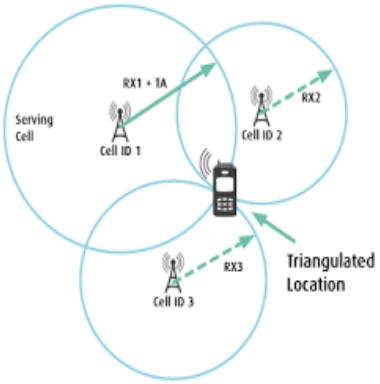


Figure 20: Positioning based on cell strength (source: `https://forums.hak5.org/index.php?/topic/32404-triangulation/`)

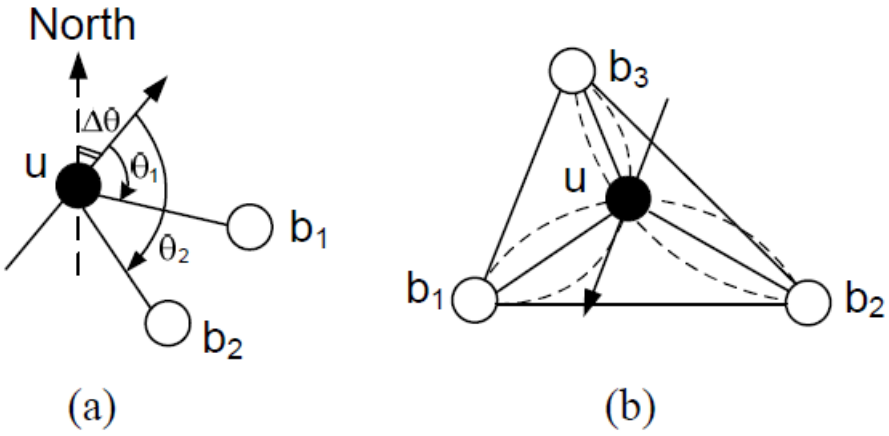


Figure 21: Triangulation in AOA localization: (a) Localization with orientation information; (b) Localization without orientation information (Singh, Shakya, & Singh, 2015)

3 Research material and methods

The final result is an Android application that provides a path to go from your actual place to the desired goal. It includes indoor and outdoor positioning and navigation. My contribution to this application has been the indoor positioning and path planning to the goal. The indoor-outdoor transition and the outdoors guidance using Google Maps have been developed by my partner, so the final application combines both parts. The indoor positioning and path planning problem are covered in this section.

3.1 Research

The final application has been designed to be as simple as possible. The resultant application shows an easy guidance system with a clear interface. The steps needed to provide that result are explained in this section. The first part was done in the computer, using MATLAB, because its better performance and easier development. Once all the theoretical parts were covered, it was exported to the mobile phone.

3.1.1 Map generation from image

Before applying any planning algorithm, you need a map that contains the basic information such as origin, goal and obstacles. The origin and the goal can be added at the end, once you have the obstacles defined in your map. There are many different ways of coding that information or creating the map. The most basic one would be to implement it by hand, relating the point coordinates with the row and column of a matrix. Obviously, that is really time consuming, but it would work.

What has been done in this thesis is taking the information of the obstacles from an image. We can do this easily with MATLAB with the command *imread*. The resulting matrix contains all the information we need. It provides a three dimensional matrix for a colored image. The first two dimensions are related with the pixels of the image, and the third one stores the color contained in each pixel.

Assuming that the floor plan given is a black and white image, the third dimension related with the colors is not needed anymore. In order to transform this three dimensional matrix to a two dimensional one, we can store the color information as if we had a white and black image. Using another MATLAB command, named *im2bw*, we can convert the original matrix to a binary image based on threshold. The resulting matrix will contain for each pixel a '0' if there is no obstacle, otherwise a '1' is stored.

After these two steps, we could compute any planning algorithm because we have

all the information needed, but it is not the best practice for our purpose. We do not really need all that information related to every single pixel. A possible way to deal with it is to decrease the resolution of the image, in other words, reduce the dimension of the matrix. For example, consider a black line in the image representing an obstacle. Even if it was a extremely thin line, it would be composed by lots of pixels. The same would happened with a corridor, or any other clear space. That would imply that the amount of nodes that the algorithm had to compute would be unaffordable.

As a example, let's take a simple map that could be a floorplan of a house, with some rooms and corridors (Figure 22a). We can create the map as it was just explained before (Figure 22b). If we zoom in the bottom left corner (Figure 23) we can observe that there are many pixel representing each obstacle. Actually, it would only be necessary to have one to compute the algorithm, so decreasing the resolution is needed.

(a) Floorplan image

(b) Map created

Figure 22: Creation of a map from an image

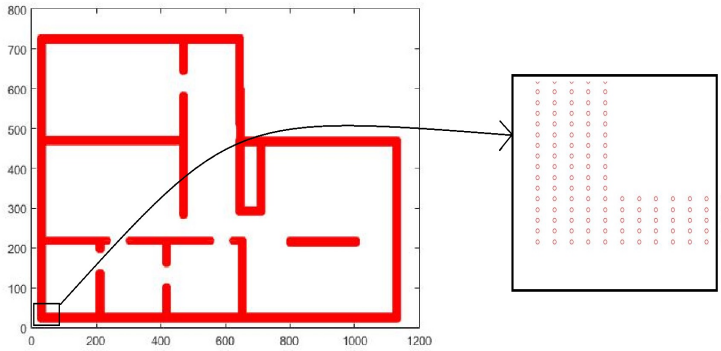
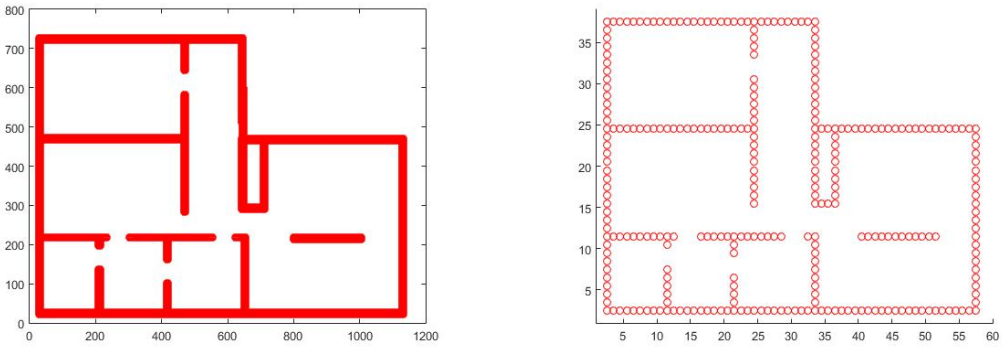


Figure 23: Zoomed section of the house map.

In order to decrease the resolution we compress some amount of pixels into a single one. The solution taken consists on creating a grid such as each cell contains some pixels, so every cell will be converted into just one pixel. Adjusting the width and

height of the cell we can optimize this process to have as much information with the least possible pixels. These values need to be tuned up for each map. In this particular example the grid was made by squared cells that contained 20x20 pixels.

Each cell will be transformed to a '1' if any of the original pixels was an obstacle. With that technique, we ensure that none of the obstacles are missing. The resulting simplified map is shown in Figure 24. Furthermore, the resulting map has a dimension of 59x38, meanwhile the original one had 1189x760. The number of pixels are extremely low compared to the original map created, so the planning algorithms will be remarkably faster.



(a) Original map (b) Simplified map

Figure 24: Simplification of a map

3.1.2 A* algorithm in MATLAB

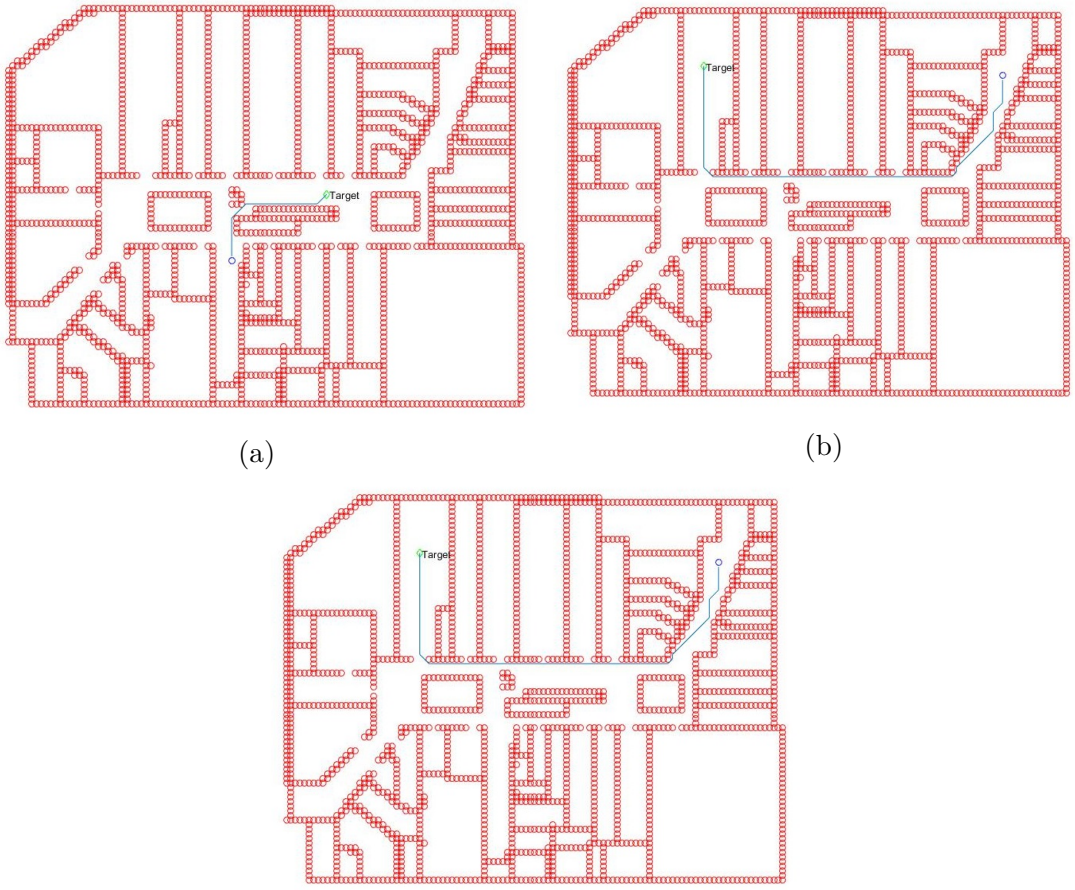
As it has already been explained in 2.1.3, A* is one of the best planning algorithms. There are some examples on the internet to apply this algorithm directly to some map, but they can not be applied to any map. As I have a custom map, I had to write the code by myself. However, that is not a problem because you know exactly where is the information you are looking for. Moreover, this algorithm had to be implemented during the subsequent development of the Android app, so it was quite easy to export it.

Following the pseudo-code showed in 2.1.3, the A* algorithm was implemented. There are some particular changes that need to be explain here. In the pseudocode it is said to make two lists, but in MATLAB there is no possibility of doing that directly, so two empty vectors were created. If I wanted to add a new element, it would be inserted at the end of the vector. In addition, another variable was needed to know how many components each vector had.

The successors of the current node were created exploring the neighbors of that node, in other words, looking for the nodes placed on the right, left, top and bottom.

After that, you have to check if any of those nodes is an obstacle or is out of the boundaries. In that case, they need to be removed as possible successors nodes. When the algorithm is finished, the path is obtained by looking back from the parent of the goal node, then the parent of the new node, and so on, until you reach the start node.

In Figure 25 you can observe some different examples of the path provided from the algorithm.



(a) (b)

(c)

Figure 25: Paths provided by the A* algorithm

Furthermore, another functionality was implemented. Once you have computed the path from your position to the goal, if your new position lies on the path the algorithm does not compute again because it realizes you are in the right path. It will allow the program to run faster and provide a better user experience, avoiding unnecessary computations. To represent that in the image, the path changes its color as it is shown in Figure 26.

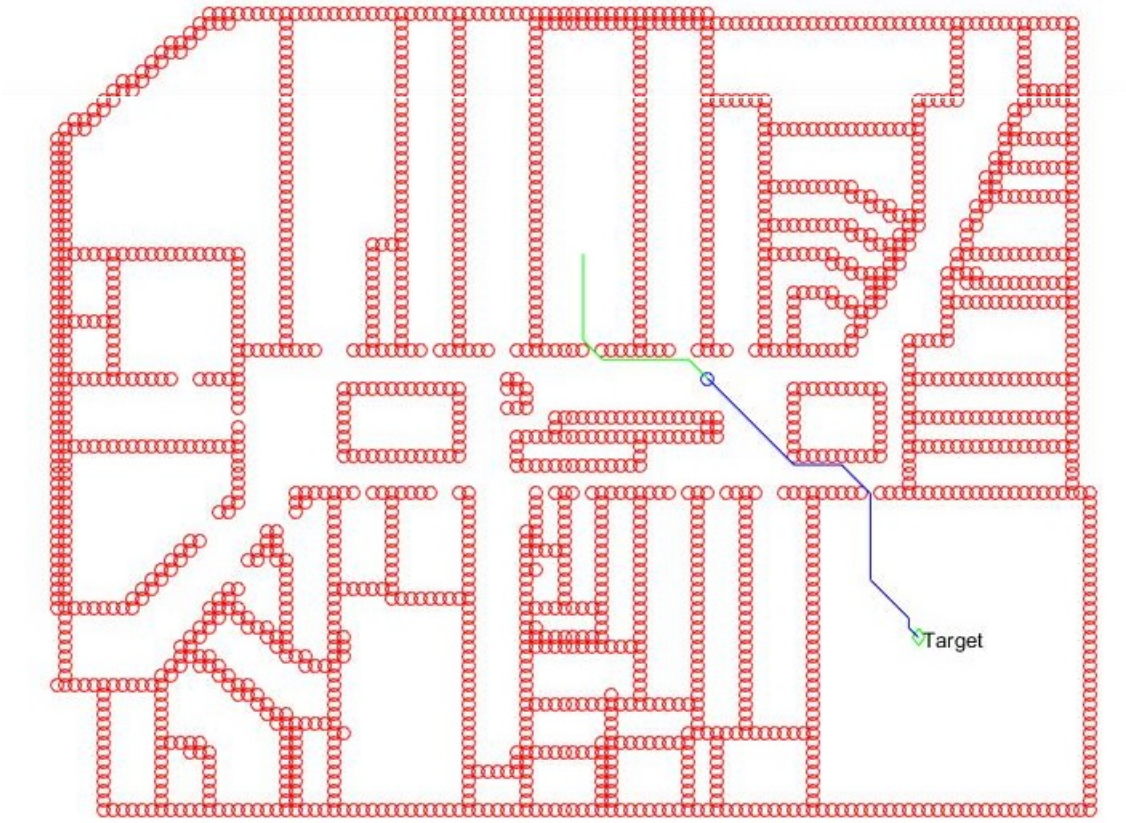


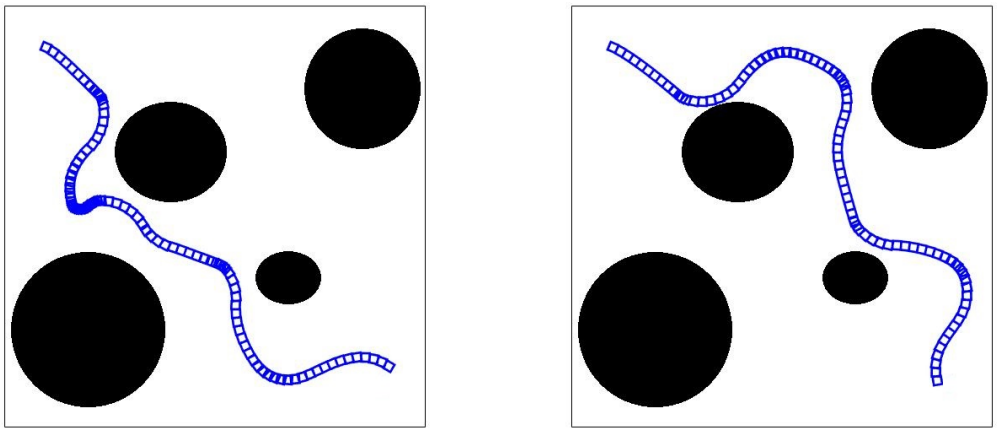
Figure 26: Zoomed section of the house map.

3.1.3 Potential field algorithm

The potential field algorithm is a common path planning algorithm to avoid certain obstacles placed in the middle of our map. However, when we have a map like the one we have for this project, this algorithm is not the best one. As it has already been explained in Section 2.1.2, this algorithm is based on generate a field through the obstacles and goal. The goal provides a attractive potential around itself and each obstacle will create a repulsive field. By following the maximum gradient starting from the origin we can obtain the path.

Once the algorithm was implemented in MATLAB, it needed tune-up for some parameters, in particular the obstacle repulsion. Depending on how much repulsion the obstacles are generating, the algorithm will be able to reach the goal or not. In Figure 27 we can see some examples of the different paths provided depending on the repulsive factor. Both of them have the same origin and goal, but the paths taken are considerably different. Nonetheless, the two paths reach the desired goal and avoiding collisions. As we can see, none of these paths are the shortest, but anyway the provided path is feasible and in some cases keep a safety distance with

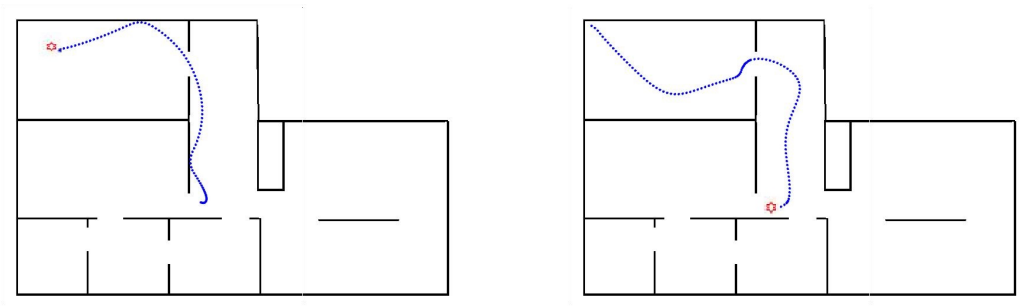
the obstacles.



(a) (b)

Figure 27: The repulsive factor in the 27a is three times bigger than the 27b

However, when we consider our floorplan the results becomes worse. The main problem here is the obstacle configuration. Instead of being circles or squares placed on the middle, now we have walls and corners. In some scenarios, the path provided do not reach the goal because it get stuck in a corner or collide with a wall in a narrow corridor. In Figure 28 some feasible paths are computed inside the floorplan (the goal is marked as a red star), but with different repulsive factor. Given a random origin and goal, the path is more likely to collide or get stuck than reach their goal. In Figure 29 we have an example in which the goal is not reached.



(a) (b)

Figure 28: Feasible paths in the floorplan using potential field algorithm

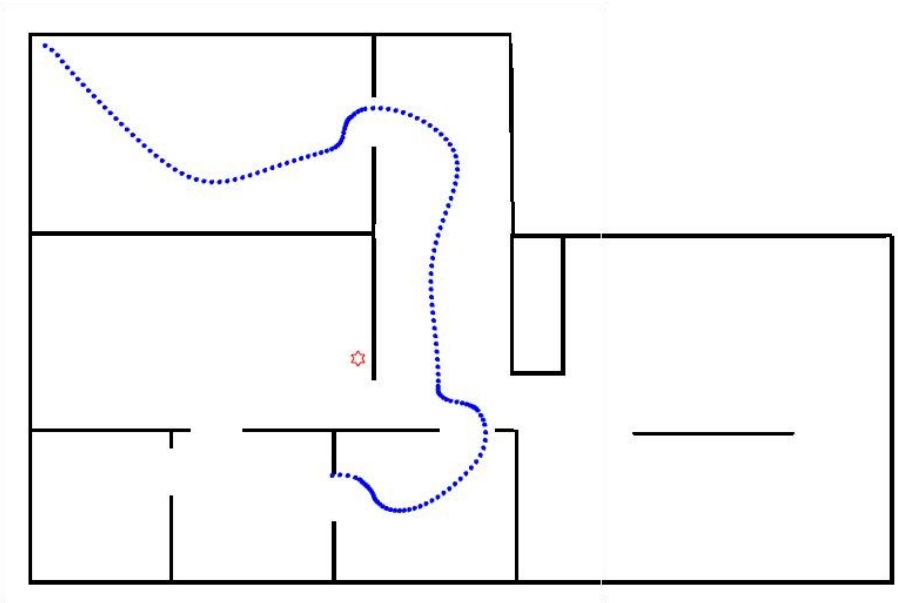


Figure 29: Not feasible path using potential field algorithm

Taking everything into consideration, the potential field algorithm is not the best one for indoor path planning. As it has already been explained, not all the paths reach their goal. Furthermore, the resultant paths are not intuitive, in the sense that it is not a normal path someone would take to go between these two points. Consequently, this algorithm does not fit the requirements our system needs. An indoor path planning system can not afford not to find a path between two points, so this algorithm can be ruled out.

3.1.4 Voronoi diagram

The Voronoi diagram and Delaunay Triangulation were obtained with MATLAB. First of all, the Delaunay's triangles were obtained. The meaning of these triangles was already been explained in 2.1.4. In order to calculate them, I used a MATLAB command named *delaunayTriangulation*. By using this class, you can also specify the constrains required with the format $DT = delaunayTriangulation(P, C)$, where P is a set of points and C the matrix that has the constrains. The resulting structure DT has some properties such as *Points*, *ConnectivityList* and *Constrains* and some methods, *isInterior* and *VoronoiDiagram*.

We can represent all the triangles contained in the full Delaunay triangulation of this floorplan with no constrains (Figure 30). The corresponding Voronoi diagram (Figure 31) provides some undesired paths, since some of them are going through

the walls. Furthermore, there are some paths outside the map boundary that are not needed either.

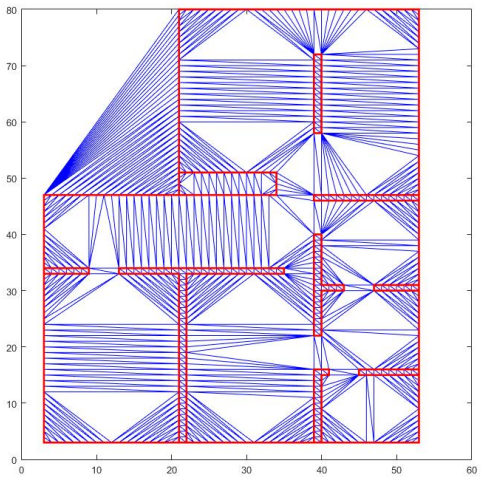


Figure 30: Delaunay triangles of a floorplan

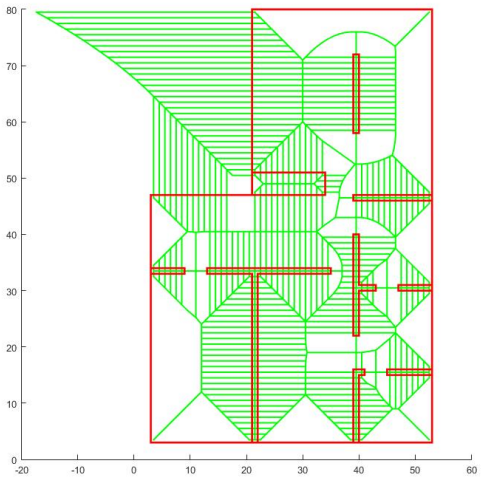


Figure 31: No constrained Voronoi Diagram

To fix that problem and try only to keep the paths that do not go through the obstacles, we have to consider the all the wall as a unique obstacle instead of a set of points. To do that, we can add some constrains to ensure that one side of the boundary triangles lies on this points. Once we have computed the new **constrained** Delaunay triangulation, we can use one of the methods given by the MATLAB class *isInterior*, which allows us to know which triangles are inside and outside the

borders. For instance, we do not care about the triangles that are outside, because we just want to obtain the Voronoi diagram from interior of the house. Moreover, those triangles that are inside the obstacles will be omitted as well. The remaining triangles are showed in Figure 32.

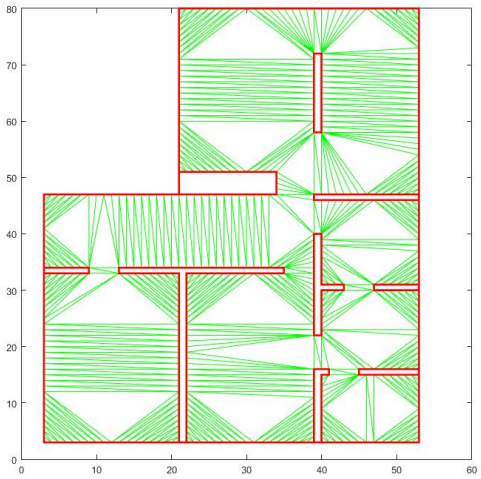


Figure 32: Interior Delaunay triangles of a floorplan

By taking the centers of the circumcircles of the remaining triangles, we can obtain the Voronoi nodes (Figure 33). Now we only have to join those points to obtain the path that maximizes the distance to the obstacles (Figure 34).

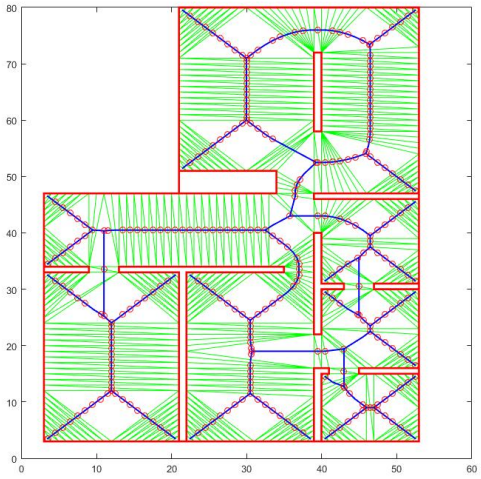


Figure 33: Voronoi diagram and nodes

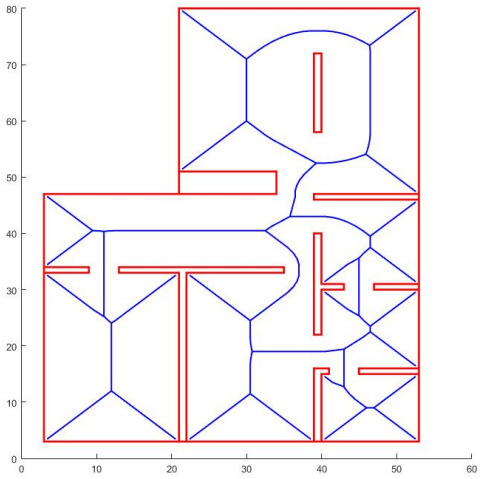


Figure 34: Voronoi diagram

If we repeat the same process with the SELLO map, where the final design will be implemented, the resulting Voronoi nodes that are needed for the planning algorithm are shown in Figure 35. These nodes have also been simplified, because of the final purpose of the project.

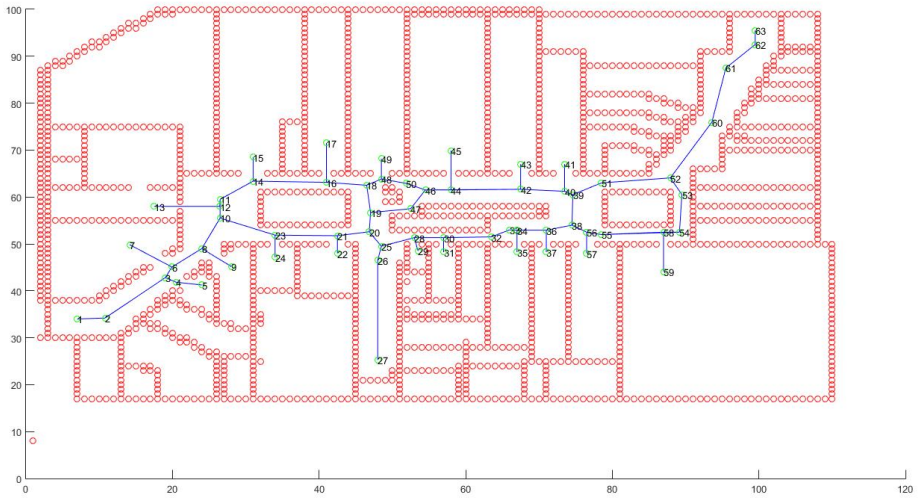


Figure 35: Voronoi nodes in SELLO

We have to keep in mind that the final purpose of the project is to guide the user to a certain part of the mall, like a specific shop or public bathrooms. For example, the points inside the area placed on the bottom right corner are useless, since the whole rectangle is representing a unique shop. This shop can be modeled using a

single node, like *node 59*. The rest of the nodes provided by the Voronoi diagram inside that shop have not been included in the final design and have been deleted manually. Logically, the same process has been repeated for the rest of the shops with the unnecessary nodes.

3.1.5 Path planning algorithm based on Voronoi nodes

In this part the actual path planning algorithm used in this project will be covered. As it has already been explained, the implemented algorithm has been A* using the Voronoi nodes. A* is a really fast computing algorithm that provides you the optimal path given a map. We have already analyzed the map and obtained the path that maximize the distance with the obstacles. So now, what we want to do is to let the algorithm compute the optimal path following those nodes provided by Voronoi.

Once we have the Voronoi nodes, we have to create a **graph** to let the path planning algorithm know which nodes are connected. For that purpose, a matrix has been created to store all this information. First of all, the nodes are listed, so we can call each node by a number. The matrix dimension is 63×3 because there are 63 nodes, and the maximum number of nodes connected to a single one is 3. In the case that a node is connected with less than three nodes, the rest of the row is filled with zeros. In the matrix, the information is given as follows: in the i row appear the nodes that are connected with it. As we can see in Figure 36, *node 1* is only connected with *node 2*, meanwhile *node 3* is connected with *nodes 2, 4* and *6*.

$$connection = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 3 & 0 \\ 2 & 4 & 6 \\ 3 & 5 & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Now that we have connected the nodes, we can code the A* algorithms as it was explained in Section 2.1.3. When the *node successors* are created, we just need to check in the connection matrix for the row associated with the current node.

There is another part that needs to be mentioned. The algorithm needs to compute the **distance between nodes**, actually between *node current* and each *node successor*, so instead of calculating that during executing time, we precomputed it and saved this data in another matrix. Furthermore, with this method we also avoid computing the same distance more than once. It is a 63×63 symmetric matrix, where the dbn_{ij} has the distance between the node i and j.

$$\mathbf{dbn} = \begin{bmatrix} dbn_{11} & dbn_{12} & dbn_{13} & \dots & dbn_{1n} \\ dbn_{21} & dbn_{22} & dbn_{23} & \dots & dbn_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ dbn_{n1} & dbn_{n2} & dbn_{n3} & \dots & dbn_{nn} \end{bmatrix}$$

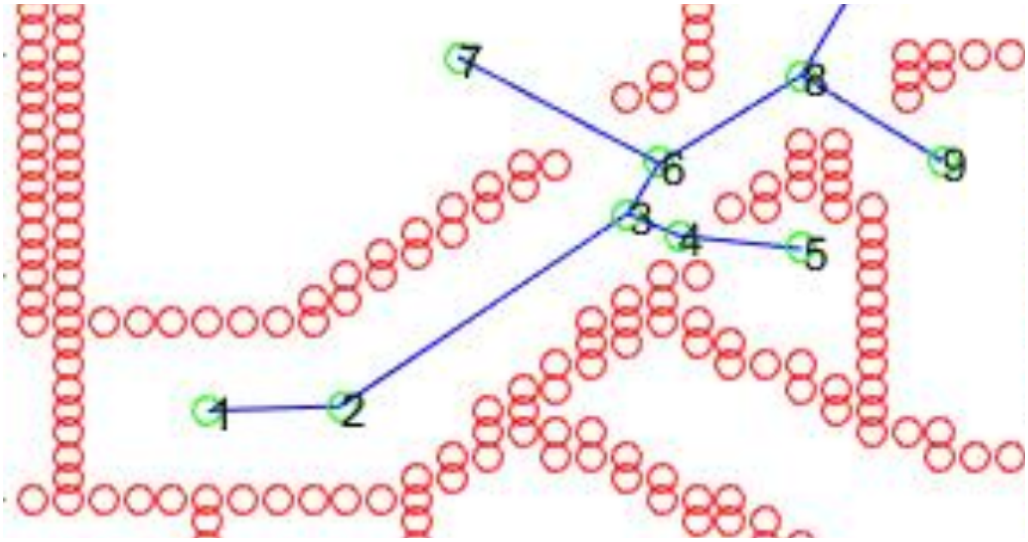


Figure 36: Part of the Voronoi diagram in SELLO

If we have the starting and goal node, the algorithm can be computed. However, the user does not have to worry about the nodes, she just wants to go from her position to a desired place. Both places do not necessarily match with an existing node. In that case, we need to know the closest node and then run the algorithm. The resulting path will be:

1. A straight line from your position to the closest Voronoi node
2. The provided path of the algorithm
3. Another straight line from the last node to the desired point.

The result can be seen in Figure 37.

In Figure 38 we can see the two different paths provided by A*, blue one, and the path planning algorithm based on Voronoi nodes, in black. The path provided by A* goes really close to the walls and obstacles, so it is not the ideal one. Moreover, as it has already been explained, the Voronoi based algorithm is much faster since it uses less nodes.

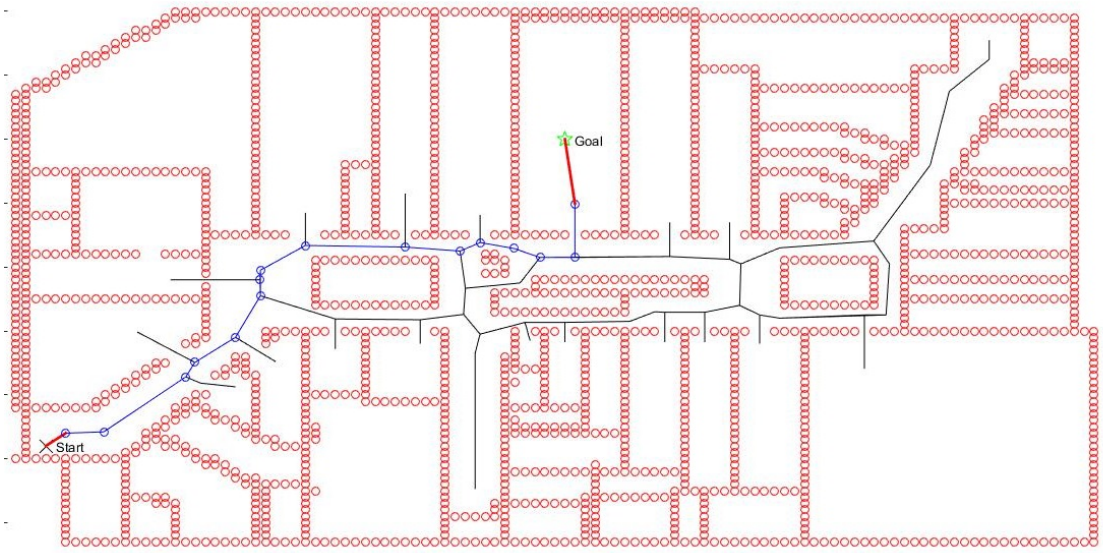


Figure 37: Example path between two points

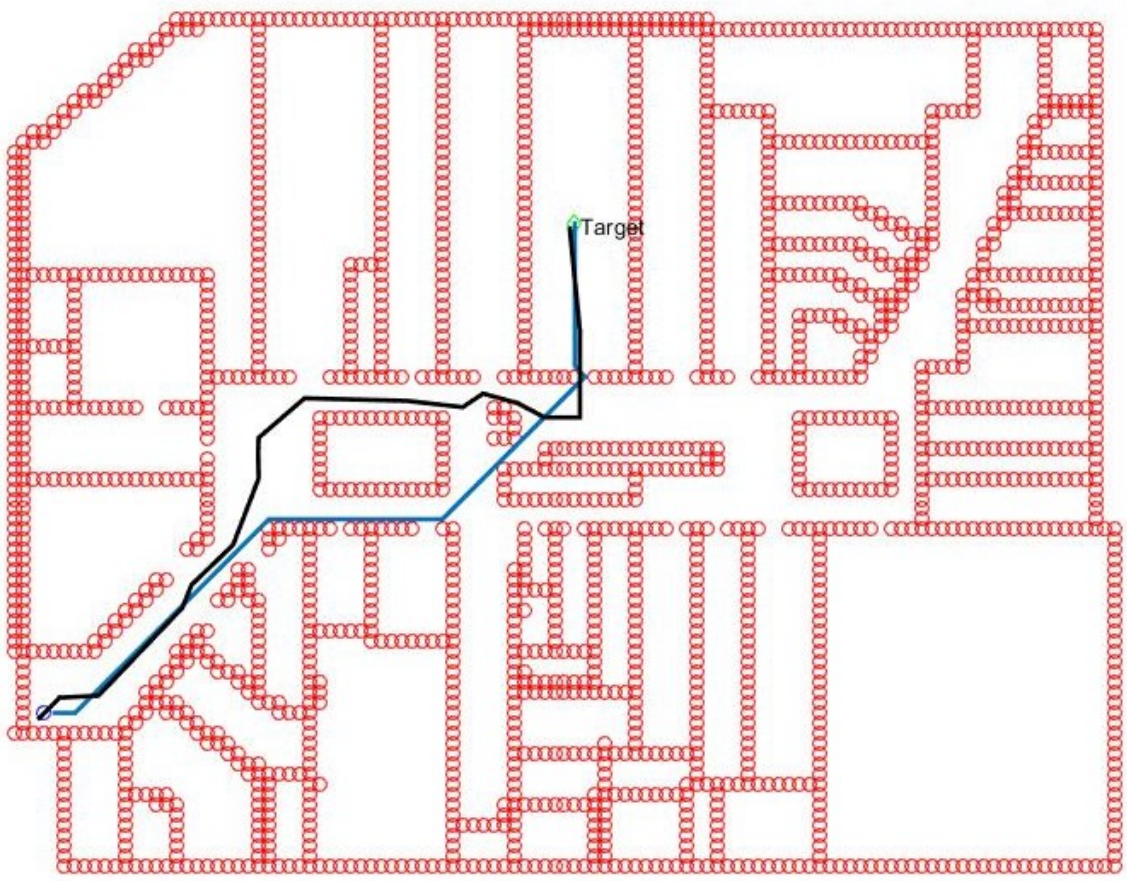


Figure 38: Comparative between the different paths provided by A* (in blue) and Voronoi+A* (black)

3.2 Implementation

In this part, the final implementation on Android will be covered. Once we have the basic algorithm running properly in MATLAB, it is easier to develop the Android application.

The application has been developed using Android Studio [2] , which is the official IDE for Android application development, based on IntelliJ IDEA[3]. This software provides smart code completion, framework-specific assistance, code templates to help you building common application features, layout editor or the possibility to create multiple APK files for your application with different features using the same project and modules. It also provides a useful debugging mode to help you find some malfunction problems.

The whole project has been installed and tested on Huawei Honor 6 (H60-L04) that runs Android 5.1.1. The results shown in this part are related to this particular mobile, so other smartphones might have different performance. Anyway, the results obtained are going to be analyzed in a general way.

Nonetheless, almost all the new smartphones should be able to run this application with an acceptable performance, since the sensors required are really basic and the application has been designed as lightweight as possible. The sensors, which the application is using are magnetometer, light sensor, GPS and gyroscope.

The **magnetometer** and **gyroscope** are needed to the indoor positioning, since the IndoorAtlas systems requires that technology to locate the user. More specific information is covered in the section 3.2.2. This Indoor Positioning System is based on the unique magnetic field that is created inside each building. Saving that magnetic field, we can later locate the user is by reading the magnetic field value in your position. As the magnetic field is measured in the three axis, the gyroscope and accelerometer are also needed to know the orientation of the smartphone.

The rest of the sensors, **GPS** and **light sensor**, are used to detect the indoor-outdoor transition. Furthermore, GPS is also the base of outdoor guidance. In outdoors environments with clear areas, GPS is the best option without a doubt.

3.2.1 Mapping a building

IndoorAtlas technology uses the smartphones to measure the unique electromagnetic field inside each building. The tools and algorithms used allow creating a map to position devices inside buildings, with no infrastructure installations needed.

[2]https://developer.android.com/studio/index.html
[3]https://www.jetbrains.com/idea

The main advantage of this system is that the Earth's magnetic field is everywhere, so it just needs to be recorded inside the desired building and, then, compare these values with the current one. With that simple idea, user can accurately be located inside solving the problem of GPS, which do not work in these environments.

This mapping part, simply means recording the sensor data until you have a good coverage of the area where positioning is required. In order to be able to map properly, minimum requirements are needed. For this part, the accelerometer, gyroscope and magnetometer are utilized. Moreover, Android version 4.3 or later is also required. These specifications are fulfilled by almost all modern smartphones, so it is not a important restriction.

IndoorAtlas uses all available sensor for positioning, even though a subset of this is usually enough to position a device. As iOS does not support WiFi scanning, it is better to use a Android device for mapping.

Pre-mapping

This is the first step in the mapping process. Once you have been register in IndoorAtlas webpage, you have to upload the floorplan of the building you want to map. This floorplan must be located in the world map, so the position of the user will be given as longitude and latitude.

Obviously, the more realistic image you have, the better results it will be provided. In this project, the first floor of a mall (SELLO) was mapped, so the actual floorplan of the building was used. It was very useful in the mapping part, since I could take some references like shops or physical structures. In Figure 39 can be seen how the floorplan is located in the world map.

Figure 39: The SELLO floorplan located in the world map

Mapping

The mapping process is done using their application *MapCreator*, available for both Android and iOS. This application will help us to map the desired place. Following the instructions given in the user interface, the user should map the selected floorplan. The mapping process is done as follows; firstly indicate on the image the path you want to map and, after that, record that path moving a steady speed. These two steps should be repeated as many times as needed to cover properly the whole area. In Figure 40 it is shown how the SELLO floorplan has been mapped.

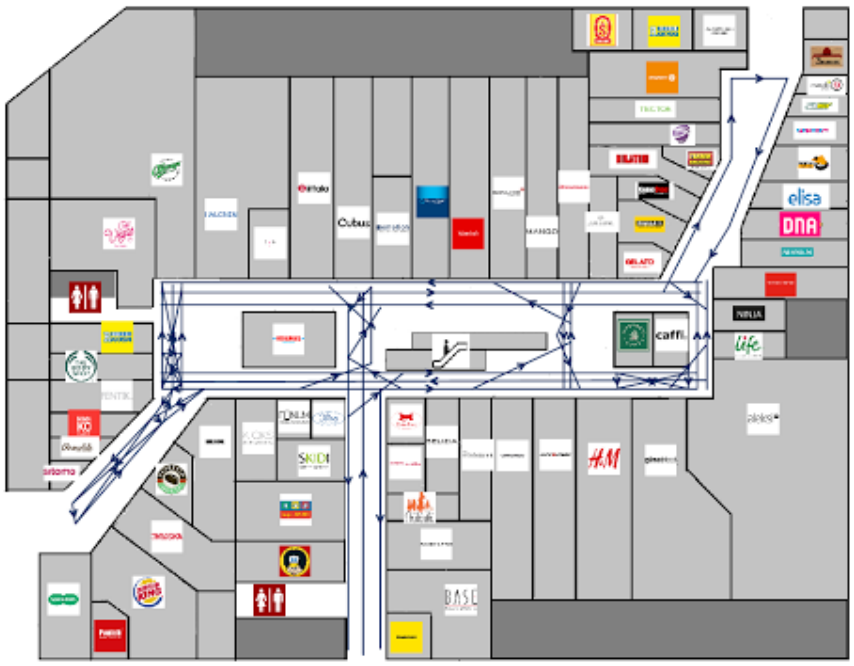


Figure 40: Paths used for mapping the floorplan

A few tips need to be considered. The recording process takes into account the starting and goal point, as well as the recording time of this path. To create an accurate map, the mapping process must be done with steady velocity, otherwise the resulting user position will not work.

Another crucial part is to connect recorded paths together to create a better map. Moreover, any gap wider than 1.5 meters will be considered as an obstacle, so for each meter of width add one parallel path to reach high level accuracy. If map has gap bigger than 1.5 meters the algorithm will not be able to calculate accurate positioning through that space. Finally, when all the required paths have been recorded, that information is uploaded to the cloud and the map will be generated.

Once the map has been created, you can start positioning your device. The positioning process will not be instantaneous, as you keep walking further in the mapped area,

the position gets more accurate. Due to the fact that magnetic field might have same values in different areas, the algorithm takes a while to locate the user. A single measure is not enough, but following measures will give the necessary information to identify which part of the magnetic data map your current, and previous, spots matches the best.

Map accuracy analysis

Once the map have been created and the positioning part is working, there is a tool for optimizing map's accuracy. It is based on *test paths*. Instead of trying to visually confirm that the positioning process match expectations, you can record test paths to be part of the map and see a quality report for your map.

Recording test paths is similar to mapping paths. Actually, the software will read magnetic field data and compare it to the mapping part. Furthermore, you can visualize the quality report that will give you and idea of how well have you map an specific area.

In Figure 41, it can be seen some different colors indicating the accuracy level in these areas. Green areas shows high accuracy, meanwhile orange and red lines tell us that these parts have not been properly mapped. When all the red and yellow lines have disappeared, as you can see in Figure 42, the mapping phase have been successfully completed.

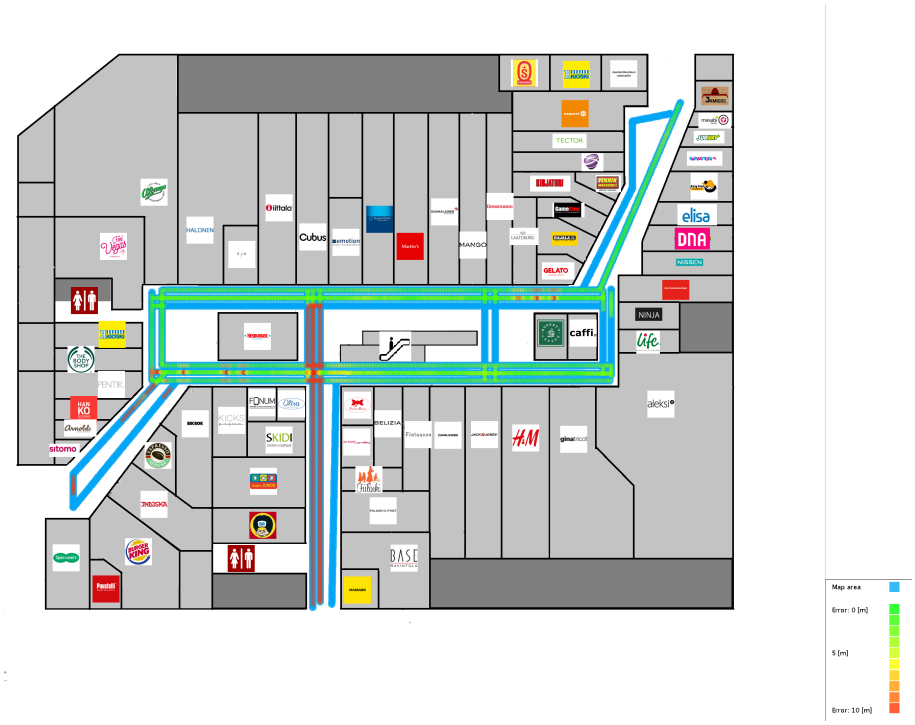


Figure 41: Quality report of the map

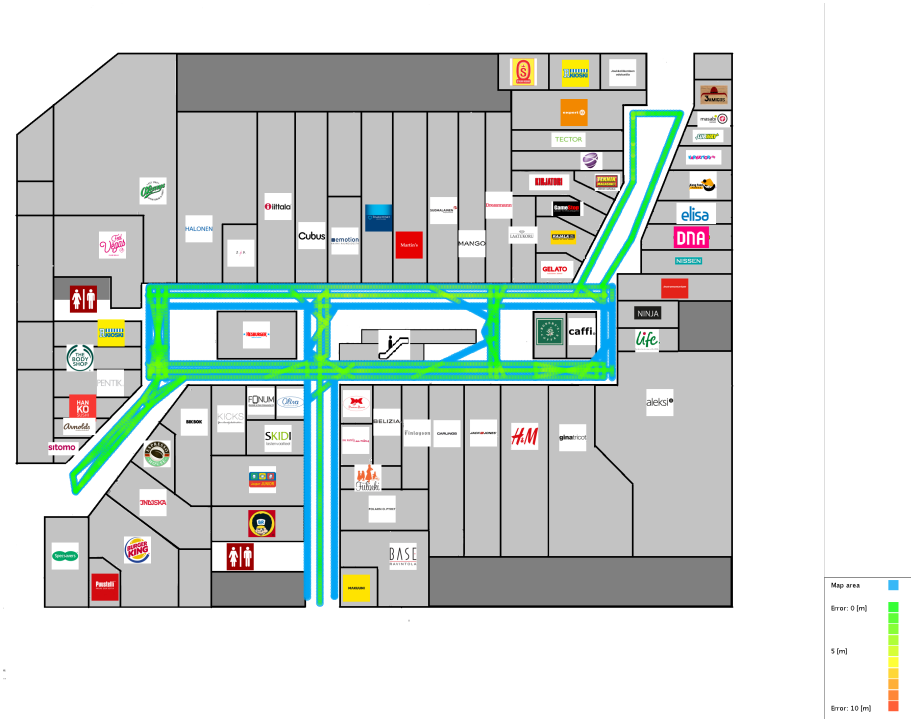


Figure 42: Quality report of the final map

This tool has been really useful to achieve a better map, because lets you know which parts need to be remapped. Even though this mapping process seems really easy, it takes some practice to create an accurate map.

Problems encountered

In the IndoorAtlas web page, there are some instructions to follow in order to create an accurate map. Actually, it seems really easy and reasonable. However, during the mapping phase, some unexpected problems appeared.

When mapping big open areas, one path is not enough. First of all, I tried to add parallel paths, but nonetheless the quality report showed low accuracy in these areas. Consequently, the position phase in these areas was inaccurate, losing the location.

Moreover, the paths should be closer than 1.5 meters, otherwise the algorithm would understand that there is an obstacle in between. To solve it, some crossed paths were needed to provide a better performance. However, in some other areas, adding extra paths induced worse results.

Finally, the final map was done by mapping the corridors with straight lines and in the tricky open areas the paths were included one by one, checking with the quality analysis if that path added useful information or it got the map worse.

Nonetheless, the first initial position takes some time to be located. When it is done,

the following tracking provides really good performance while walking through the building. The real accuracy is around 2-4 meters, which is good enough for our purpose. However, some rare cases the initial position it is not given right and the algorithm does not recover from that state. In these cases, the position process needs to be restarted.

Taking everything into consideration, the final tracking experience is acceptable, but not ideal. For the purpose of this application, there is no problem to wait around 10 or 15 seconds to be located, but in other applications this could be a concern.

3.2.2 Getting indoor position

In this part it is going to be explained how the system gets user's indoor position. As it has already been explained (Section 2.2.4), the IPS locate people or objects inside a building using different technologies, in our case, magnetic fields. These systems are used to detect and track a position, like GPS does outdoors.

This technology is based on the unique magnetic landscape produced by the Earth's magnetic field that interacts with the metallic structure of buildings. By utilizing the magnetic sensor and gyroscope, the IndoorAtlas software is able to use the magnetic field inside the building as a map to accurately position and track the user in real time.

In order to obtain the position inside the building, it is necessary to have mapped that floorplan beforehand. Once the building has properly been mapped, the IndoorAtlas SDK will provide the user's position with a reasonable accuracy, around 2 meters depending on how well the floorplan has been mapped. The details of how a building should be mapped have been explained in section 3.2.1.

Using the provided SDK in an Android project is easy following the instructions given in their web page[4]. A few steps are needed to integrate the service into your project.

1. Add the IndoorAtlas SDK as a **dependency** to your project. It is done including the following code in your *build.gradle* file.

[4]You may notice that these lines of code might change with further versions of the SDK

```
dependencies {
        compile 'com.indooratlas.android:indooratlas-android-sdk
    :2.1.2@aar'
}
repositories{
        maven {
            url "http://indooratlas-ltd.bintray.com/mvn-public"
        }
}
```

2. Add **credentials** to the Android manifest file. Every project that uses IndoorAtlas services must obtain a unique ApiKey and Secret strings. These credentials are generated in their web page, as it is shown in Figure 43.

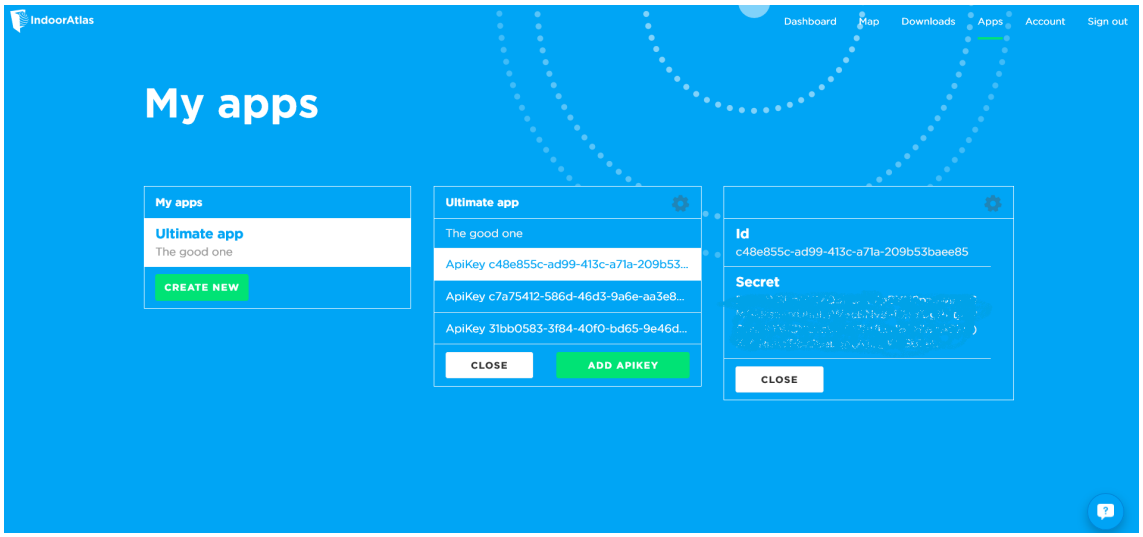


Figure 43: Creation of ApiKey and Secret strings

3. Once these strings have been created they must be added in the Android manifest file as follows:

```
<application>
    <meta-data
        android:name="com.indooratlas.android.sdk.API_KEY"
            android:value="api-key-here" />
    <meta-data
            android:name="com.indooratlas.android.sdk.API_SECRET"
            android:value="api-secret-here" />
</application>
```

When all the steps have been completed, the IndoorAtlas SDK has successfully been added to our project, so we can make use of their services. The basic methods are explained in the developer guide[5].

[5]http://docs.indooratlas.com/android/dev-guide/getting-user-location.html

First of all, on the *onCreate* method the *IALocationManager* class must be called to create a new instance of the class. It provides all the information needed for getting the position of the user inside the map.

```
private IALocationManager mIALocationManager;

protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_image_view);

        mIALocationManager = IALocationManager.create(this);
}
```

Now you can register and unregister for periodic updates of the user's current location in the *onResume()* and *onPause()* methods respectively. To start receiving the user location the method *requestLocationUpdates()* must be called. You also have to implement the *IALocationListener* and the *onLocationChanged()* callback method. In the following code, you can see where and how to call all these methods.

```
private IALocationManager mIALocationManager;

protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_image_view);

        mIALocationManager = IALocationManager.create(this);
        mIALocationManager.requestLocationUpdates(IALocationRequest.
    create(), mLocationListener);
}

private IALocationListener mLocationListener = new
    IALocationListenerSupport() {
        @Override
        public void onLocationChanged(IALocation location) {
                Log.d(TAG, "Latitude: " + location.getLatitude());
                Log.d(TAG, "Longitude: " + location.getLongitude());
        }
};
```

```
@Override
protected void onResume() {
        super.onResume();
        // starts receiving location updates
        mIALocationManager.requestLocationUpdates(IALocationRequest.
    create(), mLocationListener);
}

@Override
protected void onPause() {
        super.onPause();
        mIALocationManager.removeLocationUpdates(mLocationListener);
        unregisterReceiver(onComplete);
}
```

The location of the user is obtained with the methods *getLatitude()* and *getLongitude()* of the *IALocation* class. Using the provided classes of IndoorAtlas SDK, you can easily locate the user inside of the map adding the following code inside *onLocationChanged()* method. The *point* vector now has the position of the user related to the map.

```
IALatLng latLng = new IALatLng(location.getLatitude(), location.
    getLongitude());
PointF point = mFloorPlan.coordinateToPoint(latLng);
```

Displaying this position on a map is bit more challenging. There are many different ways to show a map and your position in Android. The one used in this project was based on **canvas** and **drawables**. The Android framework APIs provides a set of 2D-drawing APIs that allow you to render your own custom graphics onto a canvas and customize their look.

At this point, two different options were considered: draw the graphics into a View object from the layout or draw them directly to a Canvas. The first option is the best choice when you want to draw simple graphics and do not need to change dynamically. In our case, the user needs to be tracked in real-time, so the Canvas option was chosen.

Concerning the image used in the Canvas, it must be defined as a Bitmap upon which drawing will actually be performed. In our project, the Bitmap contains the map of the floorplan and the path will be painted on top of it.

The actual drawing is made in the *onDraw()* function, where the canvas is handled. There lines, points and simple figures can be painted that were later used to create the path and represent user location. Furthermore, a *Paint* class is used to describe the desired colors and styles for the drawing.

3.2.3 Reading information from file

Before starting with the code of the path planing algorithm, some more information is needed. In this part I will cover which data is required and how to obtain it. As it has already been explained, there research part was done in MATLAB. The most efficient way would be to utilize the information and data obtained in that previous work.

In the section 3.1.5 some matrices were created in order to save some information and make it easier to handle. In practice, the matrix created were:

1. **Position of the nodes** The *position* matrix contains the XY position of the nodes inside the map. This matrix will be used to paint the resulting path on the image.

$$\textbf{position} = \begin{bmatrix} p_{1x} & p_{1y} \\ p_{2x} & p_{2y} \\ \vdots & \vdots \\ p_{nx} & p_{ny} \end{bmatrix}$$

2. **Distance between nodes**. It is a $n \times n$ symmetric matrix, where the dbn_{ij} has the distance between the node i and j.

$$\textbf{dbn} = \begin{bmatrix} dbn_{11} & dbn_{12} & dbn_{13} & \dots & dbn_{1n} \\ dbn_{21} & dbn_{22} & dbn_{23} & \dots & dbn_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ dbn_{n1} & dbn_{n2} & dbn_{n3} & \dots & dbn_{nn} \end{bmatrix}$$

3. **Connection**. This matrix holds how the nodes are connected among each other. In the i row, it appears the nodes that are connected to it and if there are no three nodes connected to it, the matrix is filled with zeros.

$$connection = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 3 & 0 \\ 2 & 4 & 6 \\ 3 & 5 & 0 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

Both *dbn* and *connection* matrices are used in the A* algorithm, meanwhile the *position* matrix will be further used when the path is drawn on the map. Actually, these matrices where created directly in the computer, but they were also needed in the mobile app.

To used this information from the mobile phone, those matrices were stored in different ***.txt* files** and added to the internal storage of the mobile phone. Once these files were in the smartphone, they could be read from the application and converted again to matrices in JAVA code. The *.txt* files were written by MATLAB code.

Using this process, we avoid to create all the matrices by hand in the Android code. Moreover, this process is automatically done, so potential errors regarding typing that amount of numbers are eliminated. Another advantage of this system is the future extension to other maps. Once this process have been created, the following maps can use this system to directly send information from the computer to the mobile phone. Even that it is not a innovative solution, it fits perfectly to the purpose of the project: create a functional application that guides the user inside and outside buildings.

3.2.4 Routing algorithms

In this section, both of the routing algorithms implemented on the mobile phone will be explained. They had already been implemented on MATLAB, so this part less challenging. The first one that is covered is A* and, then, the algorithm based on Voronoi nodes.

A*

The A* algorithm was implemented in the Android application during the researching phase, but it is not in the final version. As previously mention in 3.1.2, it is easy to implement and fast computing, but the provided path is really close to the obstacles. Due to the final purpose of the app, the provided path is going to be followed by the user and his natural way of walking is to keep a safe distance with those obstacles. Therefore, this algorithm was implemented to compare its performance with the one based on Voronoi nodes. The results of this comparison are covered in 3.2.5.

The code used to compute this algorithm was based on the one created in the research part done in MATLAB, which was based on the pseudocode 2.1.3. This part should have been easier because the code was already implemented, and I had all the data needed with the information sent by the *.txt* files. However, it was more challenging than expected because in Android development there are less tools to debug code and show results than in MATLAB (e.g. a resulting matrix can be easily printed on the console, meanwhile in Android Studio you have to enter in debug mode and check the components one by one).

This is logical, since Android Studio is focused on application development rather than algorithm coding. Nonetheless, the algorithm was successfully implemented and the provided path was the same as in the research part. That previous research in the algorithm and implementation in MATLAB was really useful to the further

developing of the application.

Algorithm based on Voronoi nodes
As said in Section 3.1.5, the algorithm used is based on Voronoi nodes. With these nodes, a graph was created with the connected nodes and, then, A* computed the set of nodes that provides the required path.

The information related to the Voronoi nodes (their position and connection) was given by the *.txt* files. With all the information needed available the paths were ready to be computed. As the position of the user and the desired goal most likely will not match with one of the nodes, a new function was created to find the closest node. This function looks like this:

```
private Integer findClosestNode(PointF origin) {
    Integer node=1;
    Double dist;
    Double min=500.0;
    for (int i=0;i<nod.length;i++){
          dist=Math.sqrt((nod[i][0]-origin.x)*(nod[i][0]-origin.x)+(nod
    [i][1]-origin.y)*(nod[i][1]-origin.y));
    if (i==0)
          min=dist;
    else
          if(min>dist) {
                min = dist;
                node=i+1;
                }
          }
    return node;
    }
```

The matrix *nod* contains the position of all Voronoi nodes. With that function, the closest node to the desire point (*origin*) is returned, so the algorithm can now be run. The final path is created adding a straight line between the origin and its closest node, and the same with the goal and its closest node. In Figure 44, some example paths are shown.

3.2.5 Comparison between A* and Voronoi nodes based algorithm

Two different algorithms were used to compute the path between user's position and the desired goal. Both of them were previously applied to a map in the research part on the computer, but now it is time to run them in the mobile application and see their performance.

The first one is A*, where all points are possible nodes where the path can go through. As previously mentioned in section 3.1.2, this algorithm is remarkable fast, compared

to other planning algorithms, and provides a clear path avoiding obstacles. The main problem is that the given path might go really close to walls or any other obstacle (see Figure 25). Even though this could be a possible solution for this project, I tried to compute another algorithm to create a path that goes as far from the obstacles as possible.

(a) (b)

Figure 44: Some example paths provided by the algorithm based on Voronoi nodes

The solution finally implemented is an algorithm based on Voronoi nodes (see section 3.1.5). With this algorithm, a graph is created with the provided Voronoi nodes, so the related path will guide you through the middle of the corridors and rooms. Furthermore, this algorithm should be even faster than the A* due to the considerably less nodes that it is dealing with. The complete A* consider each point as a node, meanwhile the second algorithm only take into account some particular points as nodes.

Once both algorithms were developed in the Android application, I computed different paths to analyze their performance. The resulting executing times are displayed in the Table 1.

Table 1: Execution times of Voronoi and A* algorithms with sample paths

Elapsed time (ms)	**Path 1**	**Path 2**	**Path 3**	**Path 4**	**Path 5**
Voronoi nodes	0.57	0.80	0.37	0.54	0.62
A*	107.89	111.18	17.50	81.12	123.95
Ratio (A*/Voronoi)	190	139	47	150	200

As expected, the Voronoi nodes based algorithm provides better results than the A* due to the number of nodes. Depending on the required path planning problem considered the difference between them will be more or less. Anyway, the first one always provides appreciably better results, with and average of **145 times faster** than the second one. Obviously these data could change with other devices or

different path finding problems, but in general the second algorithm is proved to give faster results.

Nonetheless, both of them would be good enough for this system. The longest execution time of all path finding was around 100 ms. Even in that worst scenario, the user would not notice that elapsed time, because there are other aspects regarding the mobile phone that will take longer.

However, if this system was extrapolated to a remarkably bigger map, the resulting execution times could be affected. In that case, the elapsed time needed to compute the algorithm could be long enough to be noticed by the user, so the first algorithm would be better.

Even that for our purpose, both of the algorithm provides acceptably results, the Voronoi based one has been used. This choice was made not only because of the execution time, but also the provided path was not as close to the obstacles as the A*.

3.2.6 Functionality

The final application has to guide the user both indoor and outdoors, so in this section the functionality of the application will be covered. When the application is run, it detects whether the user is inside a building or not. This part has been done with an algorithm developed by my partner and provides a boolean variable with that information. Once we know that, we check the GPS position. If it is near the mall that has been mapped (*SELLO*) and the previous algorithm detects the user is inside a building, the application assume that we are inside SELLO. With that information we could have three different scenarios:

1. The user is **inside SELLO**.

   The application provides two different options: compute and display the path from your position to another indoor place of the mall or to any outdoor goal. In both cases, the IndoorAtlas software starts measuring values of the magnetic field to detect your position and it will be displayed in the map. Once the user's position has been detected with a reasonable accuracy, the path planing algorithm is run to compute the way from your position to the desired goal. If the user wants to go to a certain indoor place, she will be guided to that place. Otherwise, the indoor goal will be the exit and, then, outdoors guidance is run. The exit is preset to a specific point of the map, so the planning algorithm can be computed.

   During the indoor guidance, some features has been implemented. First of all, the path is not computed and displayed until the provided position has an **accuracy** lower than 4 meters. During the first seconds, the position given by

the IndoorAtlas SDK is not the real one, it takes around 15 seconds to provide an accurate one.

Secondly, once the path has been displayed, the blue dot tracks your position in real time. If your position goes away from the recommended path, the path planning algorithm is run again to compute a new path (**replanning**). However, if you keep going on the provided path, the path does not need to change and the path planning algorithm is not computed again.

Furthermore, if the user has chosen an outdoor goal, when the algorithm detects you have gone out, a **dialog message** appears to let the user continue with outdoors guidance. This allows the user to decide if she wants stop the guidance outside.

2. The user is **outside** In this case the user will have again the same two different options, chose an indoor or outdoor goal. In this last case, the guidance will be just outside, so Google Maps is open and the user can be guided with it. In the other case, both indoor and outdoor guidance are required, therefore is the same case as in the previous part, but in the other way around. Here, the outside guidance to the SELLO is done first, and when the user is detected to be inside SELLO another dialog message appears to continue the indoor guidance.

   Because of the researching purpose of that application, the only possible indoor goals will be inside SELLO, but it could be easily extent to other places.

3. The user is **inside a building different than SELLO** As the only building that has been mapped is SELLO, the indoor positioning in any other building is not possible. Thus, the application just inform you how to the desire place as if you were outside.

The flowchart of the functionality is shown in Figure 45. There you can see how the ideas explain before has been applied. The boxes have been filled with different colors to short them by their functionality. The green and blue boxes that are related with indoor and outdoor guidance, respectively, and the gray ones are just decision making. The rest of the colors are not directly related.

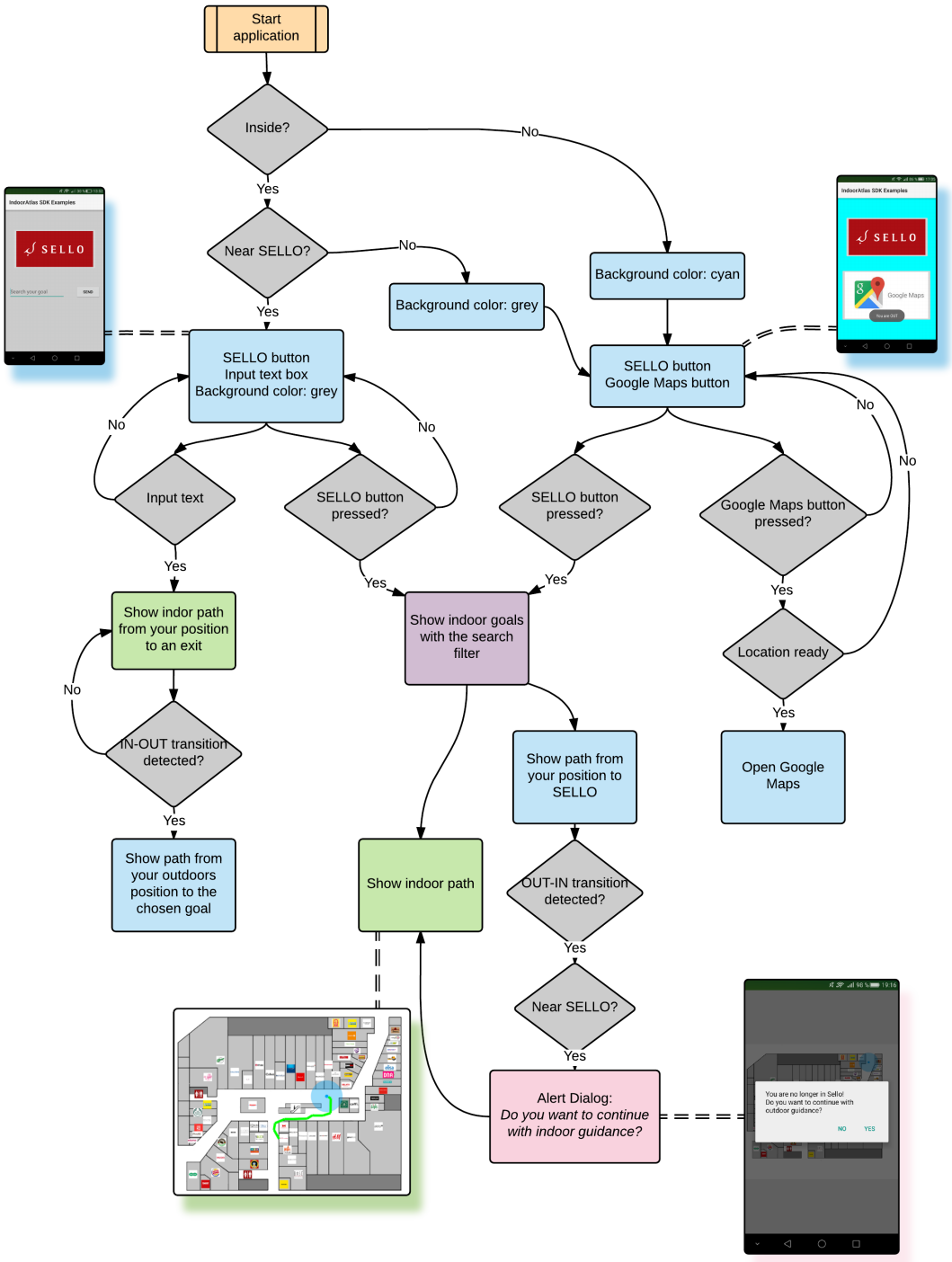


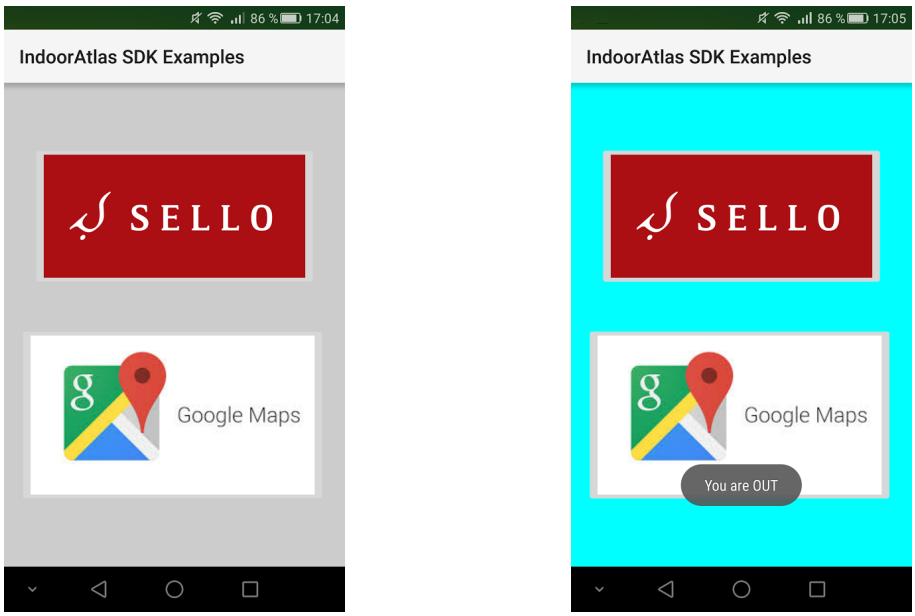
Figure 45: Flowchart of the application functionality

3.2.7 User Interface

One of the most important aspects of any application is how the user is going to interact with the product. The interface should be intuitive and simple, to provide the user a good experience with the product. The final application of this project has been designed to be as simple as possible and fulfilling the basic requirements it had. As previously mention, the application will be used in a shopping center located in Leppävaara called SELLO.

For an acceptable performance, the Android application must handle the following tasks:

- Let the user know if she is **indoors** or **outdoors**. For that purpose, the initial activity background color changes to indicate the state. Moreover, some pop up messages appear when an indoor-outdoor transition is detected. In Figure 46 it is shown how it has been done.



(a) (b)

Figure 46: In Figure 46a and 46b represent the indoor and outdoor state, respectively

- The user is able to **be guided** in all these possible options:
  1. From INSIDE SELLO to INSIDE SELLO
  2. From INSIDE SELLO to OUTSIDE SELLO
  3. From OUTSIDE SELLO to INSIDE SELLO
  4. From OUTSIDE SELLO to OUTSIDE SELLO

In the *first case*, the guidance is displayed with the recommended path you should take from your position to the chosen goal (Figure 47).

Figure 47: Indoor guidance example path

When *inside and outside* places are *combined*, the guidance is divided in two parts, one related with the indoor guidance (as the first case) and another one related with outdoors, which has been developed with Google Maps SDK.

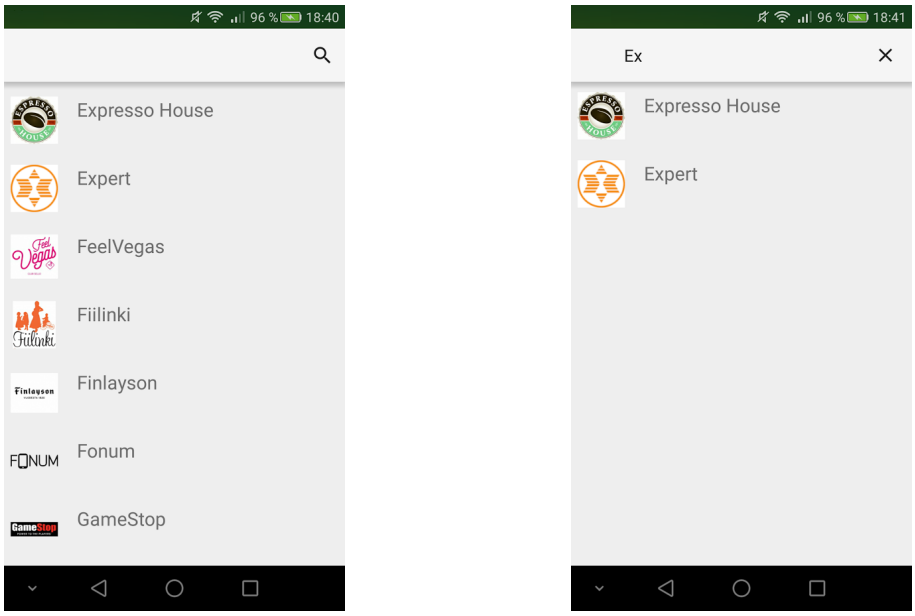
Finally, if the *origin and goal are both outside*, Google Maps application is open for that purpose. There is no need to create a new activity or app, because Google Maps provides excellent performance.

- The user can choose between a **preset indoors goals**. In order to make it easier, all the possible goals located in the first floor have been added to a list, where they can be selected. Furthermore, a search filter has also been added to provide better experience letting the user type the desired goal. If the user introduce some text, the filter only displays the items containing this text. An example is given in Figure 49.

- **Position accuracy**. The indoor positioning system takes some time to provide an accurate location of the user, so it should be displayed. The option chosen for that was to paint a circle on the estimated position (Figure 47), in such a way that the bigger radius, the less accurate the estimation is. When the accuracy reaches an acceptable value, the circle is small enough and it almost matches with the blue dot. When the accuracy level is not enough, a *progress dialog* appears to give that information. The result can be seen in Figure 49a.

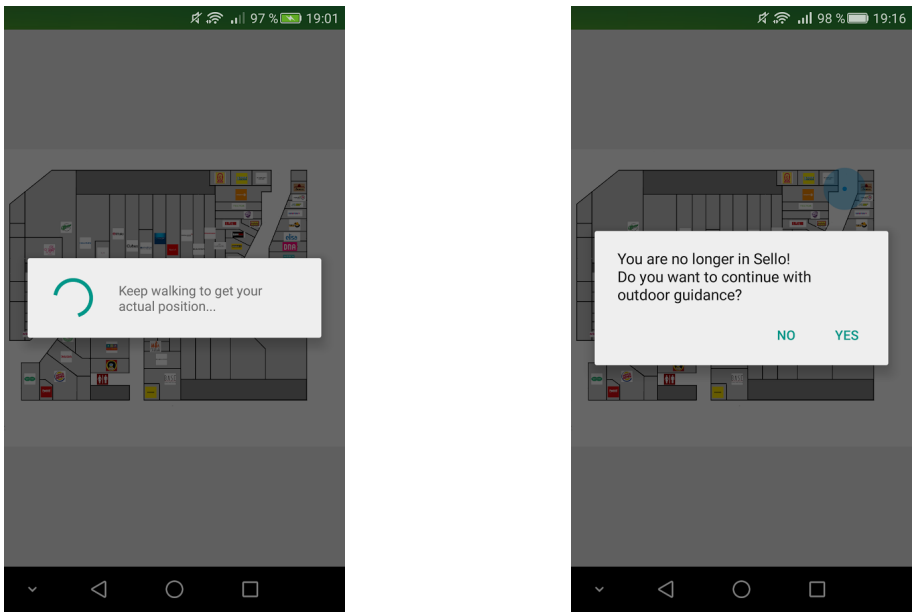
- **Transitions**. When indoor and outdoor are required, some transition is needed to indicate the user the two different phases of the process. The Figure 49b

shows the *alert dialog* that appears when the indoor-outdoor transition is detected. The user can choose between keep the outdoors guidance or finish it. This process works exactly the same in the opposite case, from outdoors to indoors.



(a) (b)

Figure 48: Search filter



(a) (b)

Figure 49: Screenshots of the application

4 Results and conclusions

4.1 Results

This Master Thesis has assisted to create an Android application that provides and **indoor and outdoor guidance**. In particular, as previously explained, the indoor positioning and path planing tasks were accomplished and solved in my thesis. The rest of the parts, indoor-outdoor transitions and outdoor guidance, have been developed by my co-worker.

This indoor positioning and path finding has been implemented in a shopping center called SELLO (see Figure 50), placed in Espoo (Finland). The users of this application could choose and indoor goal of SELLO, e.g. his favorite restaurant, and the application will guide him first, from his position to the entrance of SELLO, and then, when outdoor-indoor transition is detected the application will commute to indoor guidance. At this moment, the indoor positioning system will try to locate the user and then display the path to go to the chosen goal. The user is tracked in real time, so if she went out of the recommended path, the system would provide a new path from his current position. Otherwise, the path would remain the same.

Figure 50: Picture of the SELLO entrance

The system can be divided in four different parts:

1. **Outdoor guidance**. It has been done using Google Maps. It provides a SDK that can be added to your project, so it was the best option. The main purpose

of this part is to guide the user from its position to SELLO entrance. The outdoor position is obtained thanks to the GPS sensor.

2. **Indoor-Outdoor transition**. This part has to detect the moment when the user goes into a building or the other way around. This transition detection has been developed using the information provided by the sensor available of the mobile phone. The sensors used have been light sensor, magnetometer, cell strength and GPS. Based on some data collected, an algorithm was created to detect this transition.

3. **Indoor positioning system**. There are many different approaches to get indoor location based on ultrasonic or radio signals. However, the technology used in this thesis was based on magnetic field. The earth's magnetic field interact with the metallic structure of the buildings, creating a unique magnetic field inside. This system uses these magnetic values to create a map where users can be located.

4. **Indoor guidance**. Once the user's position is known, the application has to show a path from this location to the desired goal. To do that, some planning algorithms have been implemented and compared in order to select the one that provides the most suitable path for our purpose. The chosen one, is an algorithm based on Voronoi nodes to maximize the distance to obstacles and walls.

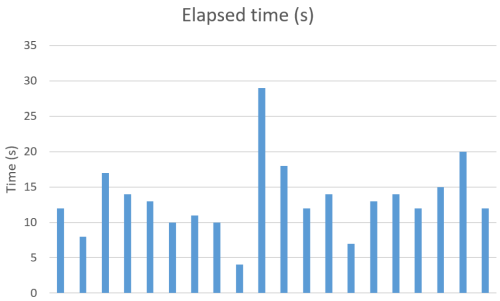
It has to be mentioned that not all of these parts were developed in this thesis. Actually, the two first have been done by my co-worker, and the last two by me. The final application is the result of both thesis combined.

4.1.1 Indoor positioning experiment

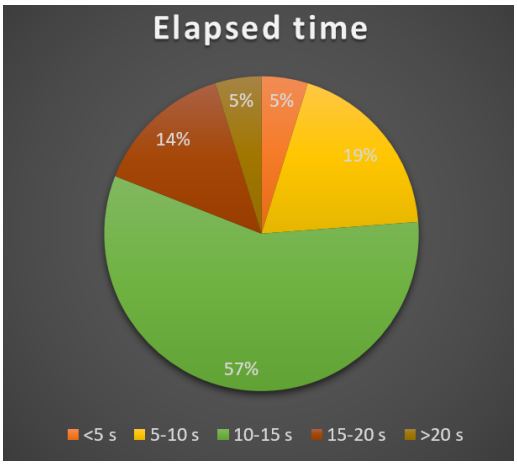
The indoor positioning process provides the current position of the user with an accuracy around 2-3 meters. For the purpose of this project, this accuracy level is enough to locate users inside the mall, but may not be accurate enough to other purposes. The main problem I encountered is that, when the software tries to locate the user for the first time, it is not instantaneous. It takes around 10-20 seconds to accurately provide user's location. The reason of that behavior is because of the magnetic field values changes a lot when you move a short distance and, also, there could be different places with the same magnetic field in that particular location. In order to find the right location, the algorithm needs to have more than one measurement of magnetic field values to provide an exact location.

Figure 51: Picture of the location experiments on SELLO

To measure this behavior and try to provide some clarifying results, an experiment was done. In this experiment (Fig 51), it was measured the elapsed time the algorithm spend to provide a location with an accuracy lower than 4 meters. The results displayed in Figure 52 show that most of the times the position is provided between 10-15 seconds. However, there are some weird cases where the exact position is provided with more than 20 seconds. Furthermore, there are some times where the given location do not match with your current position, but it is highly unlikely.



(a)



(b)

Figure 52: Indoor positioning experiment

4.2 Conclusions

The conclusions of this thesis can be summarized in the following points:

1. The initial **goals** of the project have been successfully fulfilled. The final version of the application has the desired requirements. As it has already covered, the application is able to give an exact location of the user based on magnetic field and provide a natural path from this position to the desired goal.

2. Indoor positioning systems based on **magnetic field** have been proved to be a valid alternative to conventional ones, like beacons or WLAN based systems. The accuracy reached is enough to track users or objects inside buildings. Once user's position has been detected, the system is perfectly able to track him, wherever she goes. However, when the systems starts it takes a while to detect where is the user. This *first positioning* process is not a huge problem for this project, but should be considered in other potential applications where fast positioning is a key factor.

3. Regarding path planing, **A*** has proved to be one of the best path planing algorithms due to its speed and good results provided. In spite of the rest of the path planing algorithms investigated, A* has turns out to be the most suitable for our purpose.

4. Any indoor positioning system may be really helpful in the future. There are so **many applications** where these systems can be used and help people to be guided or found in some public areas. These systems could be applied in airports, hospitals, train stations or big malls.

5. Nowadays **smartphones** provide much more information than I expected, and this information can be used for many different purposes. In this project, the magnetometer, accelerometer and gyroscope of the smartphone give enough data to know where you are. Furthermore, the indoor-outdoor transition detection, which have been developed by my co-worker, was only-based on measurements done by the mobile phone. Actually, the only hardware required to this thesis was a smartphone.

To sum up, the whole system is software-based, with the personal smartphone as the only device needed. This may allow this technology to be installed and developed in more scenarios. In general, I am happy with the results that have been obtained and with the final application developed. The application shows that this technology is able to provide a reasonable real time tracking of a user inside a building. Furthermore, this application is also able to guide you both indoors and outdoors.

5 Future lines of research

This thesis has covered two different fields, indoor positioning and path planning. Regarding path planning, so many different algorithms have been already developed and studied, so I think there is more field to study in indoor positioning systems. Nowadays, there are some attempts to provide indoor location, but none of them perform as good as GPS does outside.

Positioning systems based on **magnetic field** can be improved. The time needed for first positioning needs to be shorter for some purposes or applications. Moreover, the accuracy can also be improved and may be extended to 3D positioning. Coming back to our application, the same system could be developed in all the floors, considering the third dimension, height. With that, a completely indoor guidance systems would be done, allowing the users go to any place of the mall, not only the first floor.

Regarding other **positioning systems**, their performance needs to be improved. The existing technologies provides an accurate location in reduced areas, but when bigger indoor spaces are considered, the results become worse. Moreover, hardware devices need to be installed, like beacons or tags. In my opinion, the future systems will use existing technologies already implemented, like routers WiFi or any other system.

These indoor positioning systems will be used in many **applications**. People could be guided in airports when they are lost and need to find their boarding gate, or patients could be tracked in hospitals to know every moment where they are. Another idea could be to have more information about old people who lives alone at their home. With these systems we could know if they are at home or they have gone out.

In conclusion, these technologies may help and discover new possibilities in a near future. Many different approaches could be taken, try to improve existing methods or develop new ones. Once the technologies are proved to be efficient, a wide range of potential applications will be created.

References

Bahl, P., & Padmanabhan, V. N. (2000). Radar: An in-building rf-based user location and tracking system. In *Infocom 2000. nineteenth annual joint conference of the ieee computer and communications societies. proceedings. ieee* (Vol. 2, pp. 775–784).

Barnes, J., Rizos, C., Wang, J., Small, D., Voigt, G., Gambale, N., et al. (2002). High precision indoor and outdoor positioning using locatanet. *Positioning*, *1*(05).

Bielawa, T. M. (2005). *Position location of remote bluetooth devices* (Unpublished doctoral dissertation). Virginia Polytechnic Institute and State University.

Bill, R., Cap, C., Kofahl, M., & Mundt, T. (2009). Indoor and outdoor positioning in mobile environments—a review and some investigations on wlan-positioning. *Geographic Information Sciences*.

Bohlin, R., & Kavraki, L. E. (2000). Path planning using lazy prm. In *Robotics and automation, 2000. proceedings. icra'00. ieee international conference on* (Vol. 1, pp. 521–528).

Chawathe, S. S. (2008). Beacon placement for indoor localization using bluetooth. In *Intelligent transportation systems, 2008. itsc 2008. 11th international ieee conference on* (pp. 980–985).

Chawathe, S. S. (2009). Low-latency indoor localization using bluetooth beacons. In *Intelligent transportation systems, 2009. itsc'09. 12th international ieee conference on* (pp. 1–7).

Chew, L. P. (1989). Constrained delaunay triangulations. *Algorithmica*, *4*(1-4), 97–108.

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). Delaunay triangulations. *Computational Geometry: Algorithms and Applications*, 191–218.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). Engineering route planning algorithms. In *Algorithmics of large and complex networks* (pp. 117–139). Springer.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Djuknic, G. M., & Richton, R. E. (2001). Geolocation and assisted gps. *Computer*, *34*(2), 123–125.

Felner, A. (2011). Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm. In *Fourth annual symposium on combinatorial search.*

Figel, W. G., Shepherd, N. H., & Trammell, W. F. (1969). Vehicle location by a signal attenuation method. *Vehicular Technology, IEEE Transactions on*, *18*(3), 105–109.

Fortune, S. (1992). Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, *1*, 193–233.

Fukuju, Y., Minami, M., Morikawa, H., & Aoyama, T. (2003). Dolphin: An

autonomous indoor positioning system in ubiquitous computing environment. In *Wstfeus* (pp. 53–56).

Grewal, M. S., Weill, L. R., & Andrews, A. P. (2007). *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons.

Guibas, L. J., Knuth, D. E., & Sharir, M. (1992). Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica*, *7*(1-6), 381–413.

Hallberg, J., Nilsson, M., & Synnes, K. (2003). Positioning with bluetooth. In *Telecommunications, 2003. ict 2003. 10th international conference on* (Vol. 2, pp. 954–958).

Haverinen, J. (2015, July 14). *Generating magnetic field map for indoor positioning.* Google Patents. (US Patent 9,080,874)

Haverinen, J., & Kemppainen, A. (2009). Global indoor self-localization based on the ambient magnetic field. *Robotics and Autonomous Systems*, *57*(10), 1028–1035.

Hazas, M., & Hopper, A. (2006). Broadband ultrasonic location systems for improved indoor positioning. *Mobile Computing, IEEE Transactions on*, *5*(5), 536–547.

Hellebrandt, M., Mathar, R., & Scheibenbogen, M. (1997). Estimating position and velocity of mobiles in a cellular radio network. *Vehicular Technology, IEEE Transactions on*, *46*(1), 65–71.

Hwang, Y. K., & Ahuja, N. (1992). A potential field approach to path planning. *Robotics and Automation, IEEE Transactions on*, *8*(1), 23–32.

Kaemarungsi, K., & Krishnamurthy, P. (2004). Modeling of indoor positioning systems based on location fingerprinting. In *Infocom 2004. twenty-third annualjoint conference of the ieee computer and communications societies* (Vol. 2, pp. 1012–1022).

Kavraki, L. E., Švestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, *12*(4), 566–580.

Kirkpatrick, M., Casper, J., Ross, C., Seaton, D. S., Gellrich, C., Cutter, M., & Binnion, J. (2002). *Portable integrated indoor and outdoor positioning system and method.* Google Patents. (US Patent 6,459,989)

Koren, Y., & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and automation, 1991. proceedings., 1991 ieee international conference on* (pp. 1398–1404).

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, *27*(1), 97–109.

Koyuncu, H., & Yang, S. H. (2010). A survey of indoor positioning and object locating systems. *IJCSNS International Journal of Computer Science and Network Security*, *10*(5), 121–128.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

Leonard, J. J., & Durrant-Whyte, H. F. (1991). Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on*, *7*(3), 376–382.

Li, B., Gallagher, T., Dempster, A. G., & Rizos, C. (2012). How feasible is the use of magnetic field alone for indoor positioning? In *Indoor positioning and indoor navigation (ipin), 2012 international conference on* (pp. 1–9).

Liu, H., Darabi, H., Banerjee, P., & Liu, J. (2007). Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *37*(6), 1067–1080.

Mautz, R. (2009). Overview of current indoor positioning systems. *Geodezija ir kartografija*, *35*(1), 18–22.

Mehlhorn, K., & Sanders, P. (2008). *Algorithms and data structures: The basic toolbox*. Springer Science & Business Media.

Minami, M., Fukuju, Y., Hirasawa, K., Yokoyama, S., Mizumachi, M., Morikawa, H., & Aoyama, T. (2004). Dolphin: a practical approach for implementing a fully distributed indoor ultrasonic positioning system. In *Ubicomp 2004: Ubiquitous computing* (pp. 347–365). Springer.

Misa, T. J., & Frana, P. L. (2010). An interview with edsger w. dijkstra. *Communications of the ACM*, *53*(8), 41–47.

Moschovakis, Y. N. (2001). What is an algorithm. *Mathematics unlimited–2001 and beyond*, 919–936.

Muthukrishnan, K., Koprinkov, G., Meratnia, N., & Lijding, M. (2006). Using time-of-flight for wlan localization: feasibility study.

Ng, R., & Subrahmanian, V. S. (1992). Probabilistic logic programming. *Information and computation*, *101*(2), 150–201.

Peng, R., & Sichitiu, M. L. (2006). Angle of arrival localization for wireless sensor networks. In *Sensor and ad hoc communications and networks, 2006. secon'06. 2006 3rd annual ieee communications society on* (Vol. 1, pp. 374–382).

Prasithsangaree, P., Krishnamurthy, P., & Chrysanthis, P. K. (2002). On indoor position location with wireless lans. In *Personal, indoor and mobile radio communications, 2002. the 13th ieee international symposium on* (Vol. 2, pp. 720–724).

Priyantha, N. B. (2005). *The cricket indoor location system* (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

Riter, S., & McCoy, J. (1977). Automatic vehicle location–an overview. *IEEE Transactions on Vehicular Technology*, *26*(1).

Safadi, H. (2007). Local path planning using virtual potential field. *McGill University School of Computer Science, Tech. Rep.*

Singh, S., Shakya, R., & Singh, Y. (2015). Localization techniques in wireless sensor networks. *International Journal of Computer Science and Information Technologies*, *6*(1), 844–850.

Storms, W., Shockley, J., & Raquet, J. (2010). Magnetic field navigation in an indoor environment. In *Ubiquitous positioning indoor navigation and location based service (upinlbs), 2010* (pp. 1–10).

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing*, *1*(2), 146–160.

Yan, X., & Han, J. (2002). gspan: Graph-based substructure pattern mining. In *Data mining, 2002. icdm 2003. proceedings. 2002 ieee international conference*

on (pp. 721–724).

Yao, J., Lin, C., Xie, X., Wang, A. J., & Hung, C.-C. (2010). Path planning for virtual human motion using improved a* star algorithm. In *Information technology: New generations (itng), 2010 seventh international conference on* (pp. 1154–1158).

Yap, C. K. (1987). Ano (n logn) algorithm for the voronoi diagram of a set of simple curve segments. *Discrete & Computational Geometry*, *2*(4), 365–393.

Zandbergen, P. A., & Barbeau, S. J. (2011). Positional accuracy of assisted gps data from high-sensitivity gps-enabled mobile phones. *Journal of Navigation*, *64*(03), 381–399.

Zeimpekis, V., Giaglis, G. M., & Lekakos, G. (2002). A taxonomy of indoor and outdoor positioning techniques for mobile location services. *ACM SIGecom Exchanges*, *3*(4), 19–27.

Zeng, W., & Church, R. (2009). Finding shortest paths on real road networks: the case for a*. *International Journal of Geographical Information Science*, *23*(4), 531–543.

Zhou, R., & Hansen, E. A. (2006). Breadth-first heuristic search. *Artificial Intelligence*, *170*(4), 385–408.